

msdn magazine



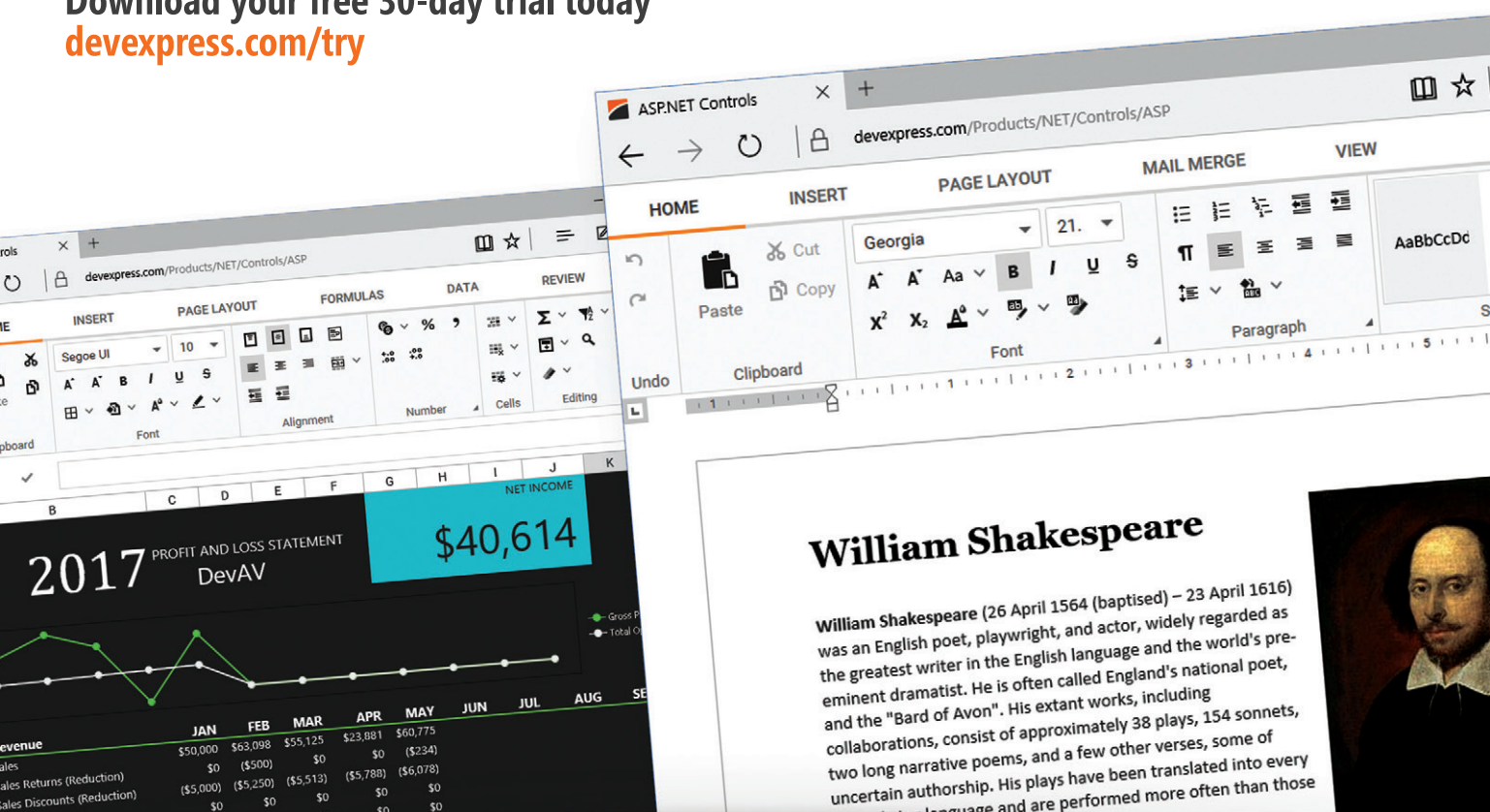
Visual Studio for Mac.....16

Office-Inspired ASP.NET & MVC Controls

Create high-impact line-of-business applications for the web with the DevExpress ASP.NET Subscription.



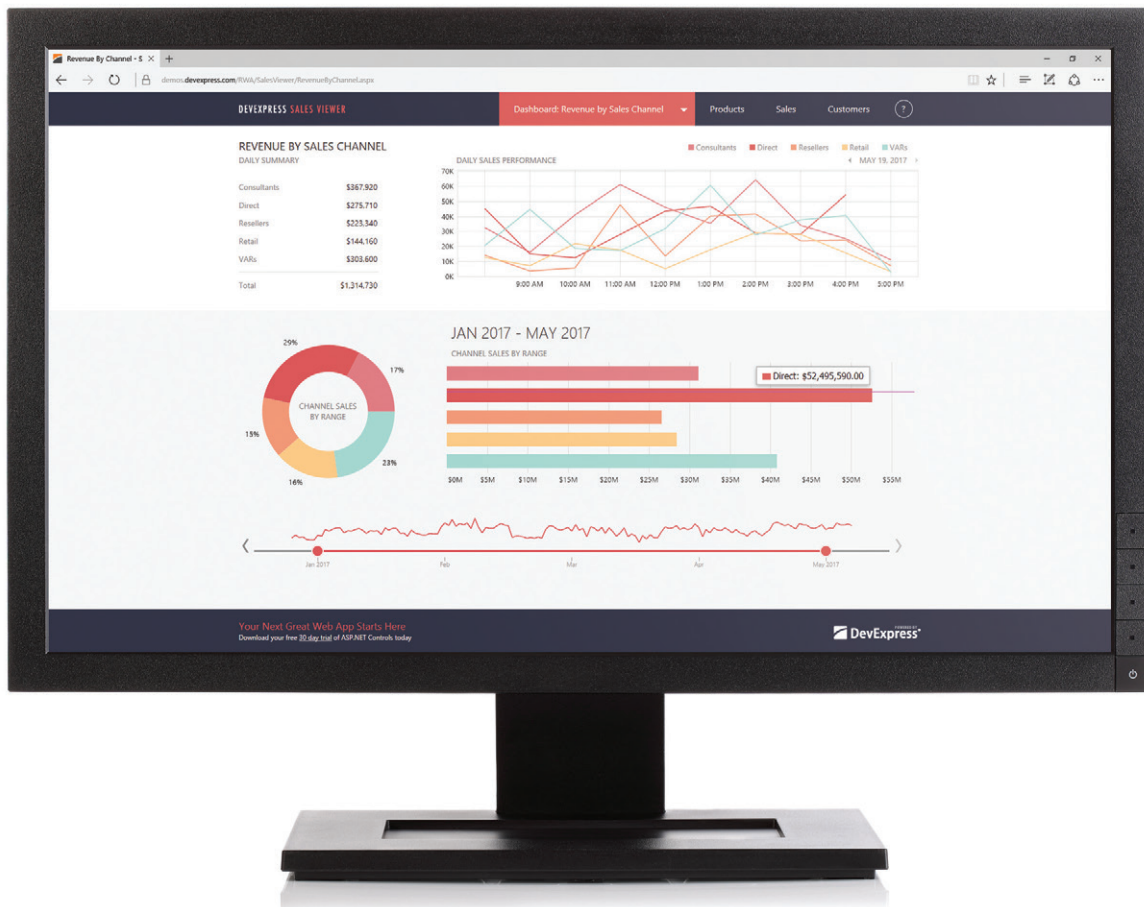
Download your free 30-day trial today
devexpress.com/try





Your Next Great Web App Starts Here

From apps that replicate the look and feel of Microsoft Office® 365, to high-impact decision support systems for your enterprise, DevExpress Web Controls for ASP.NET will help you build your best, without limits or compromise.



Download your free 30-day trial
an experience the DevExpress difference today.

devexpress.com/try

msdn

magazine



Visual Studio for Mac.....16

Code Editing and Debugging in Visual Studio for Mac Alessandro Del Sole	16
From Text to Targeted Sentiment Analysis with Cognitive Services Ashish Sahu	22
Azure Machine Learning Time Series Analysis for Anomaly Detection Dawid Borycki	30
Secure Data and Apps from Unauthorized Disclosure and Use Joe Sewell	40
Continuous Data Migration Using Visual Studio and TFS Jebarson Jebamony	54

COLUMNS

UPSTART

Creativity Under Fire
Krishnan Rangachari, page 6

ARTIFICIALLY INTELLIGENT

Exposing Machine Learning
Models from Azure ML Studio
Frank La Vigne, page 8

THE WORKING PROGRAMMER

How To Be MEAN:
Angular Forms
Ted Neward, page 12

CUTTING EDGE

Guidelines for ASP.NET MVC
Core Views
Dino Esposito, page 62

TEST RUN

Kernel Logistic Regression
Using C#
James McCaffrey, page 66

DON'T GET ME STARTED

Duct Tape
David Platt, page 72



Infragistics Ultimate 17.2

Productivity Tools & Fast Performing UI Controls for Quickly Building Web, Desktop, & Mobile Apps

Includes 100+ beautifully styled, high-performance grids, charts & other UI controls, plus rapid prototyping, wire-framing & visual configuration tooling!

Angular | JavaScript / HML5 | ASP.NET MVC | Windows Forms | WPF | Xamarin

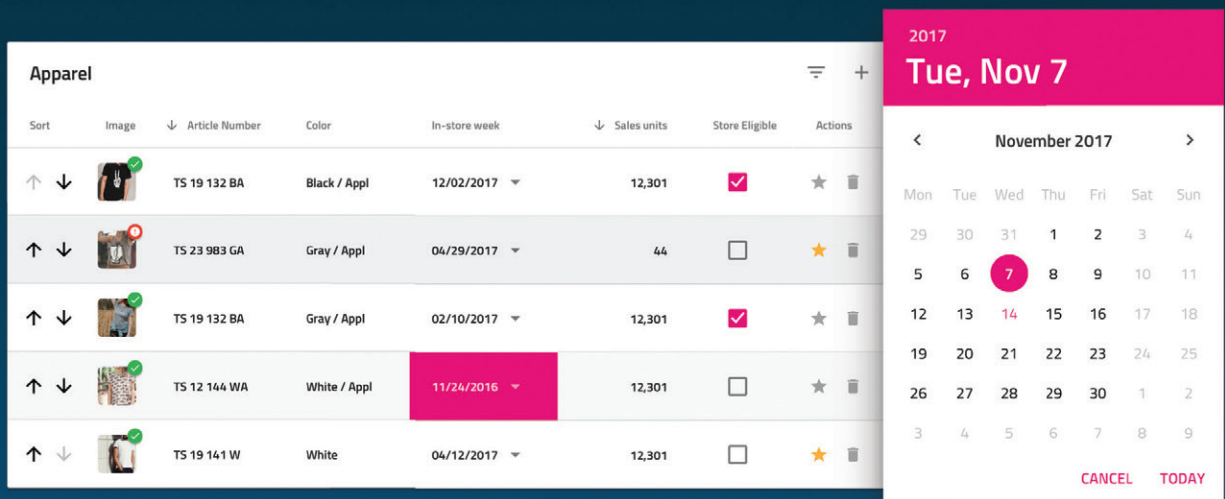
Download a free trial at
Infragistics.com/Ulimate



Featuring

Ignite UI

A complete UI component library for building high-performance, data rich web applications



- ✓ Create beautiful, touch-first, responsive desktop & mobile web apps with over 100 JavaScript / HTML5, MVC & **Angular components**.
- ✓ Speed up development time with responsive layout, powerful data binding, cross-browser compatibility, WYSIWYG page design, & built-in-themes.
- ✓ Our easy to use Angular components have no 3rd party dependencies, a tiny footprint, and easy-to-use API.
- ✓ The Ignite UI **Angular Data Grid** enables you to quickly bind data with little coding - including features like sorting, filtering, paging, movable columns, templates and more!

Download a free trial of Ignite UI at: **Infragistics.com/ignite-ui**

To speak with our sales team or request a product demo call: 1.800.321.8588

General Manager Jeff Sandquist

Director Dan Fernandez

Editorial Director Mohammad Al-Sabt mmeditor@microsoft.com

Site Manager Kent Sharkey

Editorial Director, Enterprise Computing Group Scott Bekker

Editor in Chief Michael Desmond

Features Editor Sharon Terdeman

Group Managing Editor Wendy Hernandez

Senior Contributing Editor Dr. James McCaffrey

Contributing Editors Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt

Vice President, Art and Brand Design Scott Shultz

Art Director Joshua Gould



President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

ART STAFF

Creative Director Jeffrey Langkau
Associate Creative Director Scott Rovin
Senior Art Director Deirdre Hoffman
Art Director Michele Singh
Art Director Chris Main
Senior Graphic Designer Alan Tao
Senior Web Designer Martin Peace

PRODUCTION STAFF

Print Production Manager Peter B. Weller
Print Production Coordinator Lee Alexander

ADVERTISING AND SALES

Chief Revenue Officer Dan LaBianca
Regional Sales Manager Christopher Kourtoglou
Advertising Sales Associate Tanya Egenolf

ONLINE/DIGITAL MEDIA

Vice President, Digital Strategy Becky Nagel
Senior Site Producer, News Kurt Mackie
Senior Site Producer Gladys Rama
Site Producer Chris Paoli
Site Producer, News David Ramel
Director, Site Administration Shane Lee
Front-End Developer Anya Smolinski
Junior Front-End Developer Casey Rysavy
Office Manager & Site Assoc. James Bowling

LEAD SERVICES

Vice President, Lead Services Michele Imgrund
Senior Director, Audience Development & Data Procurement Annette Levee
Director, Audience Development & Lead Generation Marketing Irene Fincher
Director, Client Services & Webinar Production Tracy Cook
Director, Lead Generation Marketing Eric Yoshizuru
Director, Custom Assets & Client Services Mallory Bastionell
Senior Program Manager, Client Services & Webinar Production Chris Flack
Project Manager, Lead Generation Marketing Mahal Ramos

ENTERPRISE COMPUTING GROUP EVENTS

Vice President, Events Brent Sutton
Senior Director, Operations Sara Ross
Senior Director, Event Marketing Merikay Marzoni
Events Sponsorship Sales Danna Vedder
Senior Manager, Events Danielle Potts
Coordinator, Event Marketing Michelle Cheng
Coordinator, Event Marketing Chantelle Wallace



Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Chief Financial Officer
Craig Rucker

Chief Technology Officer
Erik A. Lindgren

Executive Vice President
Michael J. Valenti

Chairman of the Board
Jeffrey S. Klein

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o MSDN Magazine, 4 Venture, Suite 150, Irvine, CA 92618.

LEGAL DISCLAIMER The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International
Phone: 212-221-9595
E-mail: 1105reprints@parsintl.com
Web: 1105Reprints.com

LIST RENTAL This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jloug@meritdirect.com; Web: meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. E-mail: To e-mail any member of the staff, please use the following form: FirstInitialLastName@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311
The opinions expressed within the articles and other contents herein do not necessarily express those of the publisher.



WORLD-CLASS MOBILE RECOGNITION SDK

Quickly and accurately extract data from any document or image with LEADTOOLS. Developers using the SDK easily add advanced OCR, Barcode, Forms, Driver's License, Passport, Check, and Credit Card recognition functionality into their applications.

Download the SDK today and start developing with the best in:



RELIABILITY



ACCURACY



SPEED



Fully-functional sample apps built with LEADTOOLS are also available for download from Google Play, App Store, and Microsoft Store.





Get Smart

Artificial intelligence (AI). Machine learning (ML). Data science. Terms like these have been cropping up more and more lately in *MSDN Magazine*, and for good reason. The confluence of cloud computing, Big Data and advances in deep learning algorithms has opened new horizons for developers, as tools emerge to wire up intelligent systems that drive insight, improve decision making and speed execution.

In July, we published a package of four feature articles focused on ML, including Wee Hyong Tok's "Doing Data Science and AI with SQL Server" (msdn.com/magazine/mt784663) and James McCaffrey's "Introduction to the Microsoft CNTK v2.0 Library" (msdn.com/magazine/mt784662). Microsoft Cognitive Services has been the subject of exploration, from contributors like Alessandro Del Sole ("Face and Emotion Recognition in Xamarin.Forms with Microsoft Cognitive Services" msdn.com/magazine/mt742868) and in this issue, Ashish Sahu's "From Text to Targeted Sentiment Analysis with Cognitive Services."

To help readers keep up, *MSDN Magazine* last month debuted a new column focused on AI and ML, called Artificially Intelligent.

The amount of activity in the AI/ML space is both impressive and maddening, as evidenced by announcements at the Microsoft Ignite conference in September. Microsoft Azure Machine Learning Studio, released in 2015, has emerged as the go-to tool for rapid development and system interaction via Web services. But the drive for more control over both algorithms and frameworks, combined with limitations in model deployments, prompted Microsoft to release new solutions, like Azure Machine Learning with

its Workbench client, Experimentation Service and Model Management Service. At Ignite Microsoft also announced Visual Studio Code Tools for AI alongside updates to Cognitive Services.

The tools are evolving fast, opening powerful new capabilities even as they force developers to retool and retrain on-the-fly. To quote Ron Burgundy in the movie "Anchorman," after he discovers that his dog Baxter has eaten an entire wheel of cheese: "I'm not even mad. That's amazing."

To help readers keep up, *MSDN Magazine* last month debuted a new column focused on AI and ML, called Artificially Intelligent (msdn.com/magazine/mt826348). The author, Frank La Vigne, has been writing the Modern Apps column focused on Universal Windows Platform (UWP) development, but now shifts his sights to AI. It's an arena he's been active in since he attended the Data Science and Machine Learning Summit in 2016 on the Microsoft campus. He came away convinced that Microsoft was "planning to integrate AI into every facet of the company," and immediately immersed himself in the space.

La Vigne says Microsoft's AI strategy is shaped by its historical focus on putting powerful tools into the hands of users.

"I think the overall theme of Microsoft's products through the years has been democratization, whether that's making developer tools or office productivity tools. The same holds true here, making machine learning and AI more accessible to developers," La Vigne says, singling out the Cognitive Services support for computer vision and natural language processing.

As for what it takes for developers to thrive in the AI/ML space, La Vigne says it comes down to a commitment to constant learning.

"Microsoft really has to sell developers and data engineers that data science, AI and ML is not some big, scary, hyper-nerd technology. There are corners where that is true, but this field is open to anyone that is willing to learn," La Vigne says. "In fact, anyone who is already a developer or engineer clearly already has the core capabilities to be successful in this field. All people need to do is level up some of their skills and add new ones."

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2017 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.

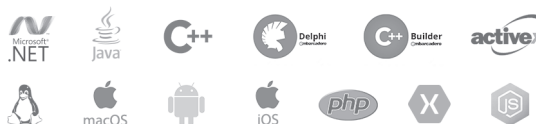
We didn't invent the Internet...

...but our components help you power the apps that bring it to business.



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
AS2, AS4, EDI/X12, OFTP ...
- **Credit Card Processing**
Authorize.Net, ACH, 3-D Secure ...
- **Shipping & Tracking**
FedEx, UPS, USPS ...
- **Accounting & Banking**
QuickBooks, OFX, SAP ...
- **Internet Business**
Amazon, PayPal, Google ...
- **Internet Protocols**
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**
SSH, SFTP, SSL, Certificates ...
- **Encryption & Certificates**
X.509, OpenPGP, SHA, S/MIME ...
- **Network Management**
SNMP, MIB, LDAP, Monitoring ...
- **Compression**
Zip, Gzip, Jar, AES, 7Zip ...



Our **Red Carpet Subscription** includes all product lines + updates for one year.

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. For more than 20 years, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

Creativity Under Fire

Recently, a reader e-mailed me about a simple software engineering project that had become too complex. He felt that he didn't have the skills or resources to handle it himself anymore, and he didn't know what to do.

In such a situation, there are really two problems: how to escape from feeling overwhelmed by a complex project, and how to be creative enough to find a solution. Let's tackle the two problems separately.

When it comes to feeling overwhelmed, here are 10 strategies I rely on to free myself:

1. **Draft It:** I create a preliminary, rough version of my deliverable, based on what I know. Then I get it ready for review from a low-risk colleague, like a trusted peer. I challenge myself to get this done within 20 percent of the time it would take me to do the whole project.
2. **Redefine It:** I may be overwhelmed because the project's goal is too vague, or I'm trying to do too much. So I make the goal simultaneously more concrete and simpler.
3. **Leverage It:** I reuse as much as I can from other people's work. I focus only on where I can add value with my own unique additions and insights. If it's not unique, I copy from others.
4. **Credit It:** I invite another person to collaborate on the project with me, and I give them most of the credit. The more credit I give to others, the less pressure I'll feel to be the project's lone-star savior, though I still give the project the very best I can. Interestingly, I've found that when I collaborate with others and share credit, we don't *divide* the resulting rewards; we *multiply* them.
5. **Atomize It:** I break the project into its fundamental, most uncomfortable, smallest parts, then I pick one, high-impact part to focus on. Fulfillment in a project doesn't come merely from "getting things done." It comes from working on its important, uncomfortable parts non-compulsively.
6. **Fake It:** Sometimes, the overwhelmed feeling is just a trick of the mind. I ignore it and act as if I were an amazing engineer who knows exactly what to do. Then I go do it.
7. **Game It:** I change the rules, sometimes to an extreme. Instead of trying to do a project in four weeks, I ask myself how I can get the whole thing done in four hours. (Have you ever finished an overwhelming project at the very last minute? Well, you don't have to wait until deadline to channel that laser-like focus; you can train yourself to do it anytime.)
8. **Skip It:** I might be subjecting myself to a futile exercise in pointlessness. If I suspect this is the case, I simply stop working on the project and move on.

9. **Partition It:** I may have said "yes" to work that's not mine to do. So, I identify only the parts of the project that are mine, do those, and reassign, delegate, eliminate or disown the rest.
10. **Review It:** I come up with two or three solutions in my mind, then present them to my colleagues and ask for their thoughts. Through their iterative feedback, I crowdfund my solution.

Interestingly, I've found that
when I collaborate with others
and share credit, we don't
divide the resulting rewards;
we multiply them.

Now, let's discuss the second problem: creativity. Perhaps you're comfortable with fixing bugs, but don't feel as comfortable with large-scale, open-ended problems. How can you be creative enough to craft architectural designs, devise technical strategy, and make proposals that currently feel out of your reach? There are three strategies I use:

1. **Transfer It:** When I express my creativity outside work—whether through singing, dancing, acting, improv or writing—I find that I'm more creative at work. Over time, my mind learns to believe that I'm creative.
2. **Detach It:** When I feel stuck, sometimes what holds me back is wanting a particular result. Even wanting a little bit of recognition can dampen my creativity, because it introduces fear—and fear and creativity can't co-exist. So, I redefine my goal to only my input (that is, the work I do) on the project; the output (that is, what praise I get) is no longer in the picture. It also helps to reframe the project in terms of the value I'd like to give others. Whether others indeed derive value is out of my control.
3. **Sandbox It:** I focus on being extra-creative in an area where I'm already deeply comfortable. This could be a side project, a private skunkworks project or even a non-work project. This gives me a safe space to push my limits and build my own sense of creative comfort. Over time, I can start to expand beyond these limits. ■

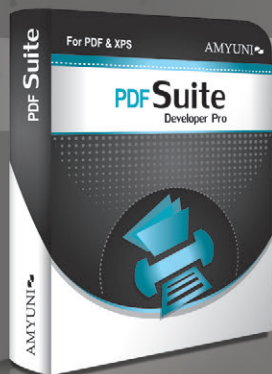
KRISHNAN RANGACHARI helps engineering managers have more impact. Visit RadicalShifts.com for his free course.



Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

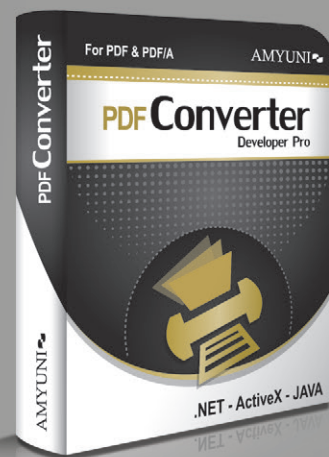
NEW
v5.5

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module



Complete Suite of Accurate PDF Components

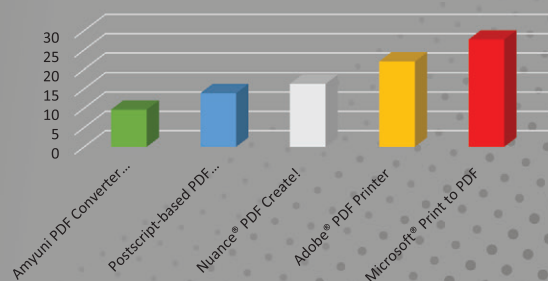
- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment



High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



Other Developer Components from Amyuni®

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need



AMYUNI 
Technologies

USA and Canada

Toll Free: 1866 926 9864

Support: 514 868 9227

sales@amyuni.com

Europe

UK: 0800-015-4682

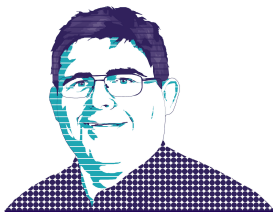
Germany: 0800-183-0923

France: 0800-911-248

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.

All development tools available at

www.amyuni.com



Exposing Machine Learning Models from Azure Machine Learning Studio

One of the key drivers of innovation in the machine learning (ML) and artificial intelligence (AI) space has been the fact that much of the results of cutting-edge research has “left the lab” and is available to the public. Services such as Cortana and Skype Translate that rely on years of work in speech recognition and linguistic analysis are in the hands of users worldwide.

In my previous column (msdn.microsoft.com/mt826348), I demonstrated how to create an ML model in Azure Machine Learning Studio. Using a sample experiment as a starting point, I created a model that predicted whether a flight would be canceled with an accuracy rate of slightly greater than 80 percent. While this model might be useful on its own, the full transformative power of AI would be realized if I made this model available to more than just the data scientists on my team. What if this predictive model were exposed to business analysts? What if it was made easy for them to consume the predictive model from within Excel? What if the model was exposed to app developers to embed into any number of applications?

In this article, I'll demonstrate how to expose ML models created in Machine Learning Studio to non-data-scientist users.

Fortunately, Machine Learning Studio makes it straightforward to expose ML models via a Web service.

Creating a Web Service in Machine Learning Studio

Fortunately, Machine Learning Studio makes it straightforward to expose ML models via a Web service. Browse over to Machine Learning Studio at studio.azureml.net. If you followed the steps in my previous column, please open up the Binary Classification: Flight

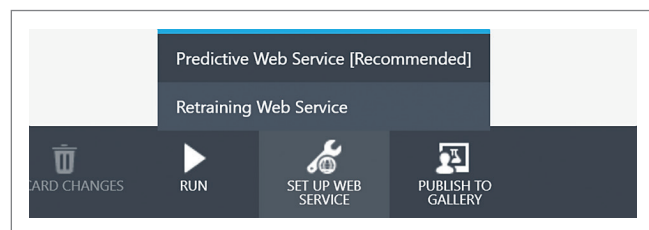


Figure 1 The Set Up Web Service Context Menu in Azure Machine Learning Studio

delay prediction experiment. If not, go ahead and follow the steps I outlined last month to create the experiment.

Once loaded, there should be a button at the bottom of the Web page called Set Up Web Service. Click on it and click on the Predictive Web Service [Recommended] option in the context menu that appears (see **Figure 1**). If the option to create a Predictive Web Service is greyed out, then you might have to run the experiment again. Click the Run button to the immediate left of the Set Up Web Service button as shown in **Figure 1**.

Once you click the menu option, the dialog in **Figure 2** will appear. Machine Learning Studio will prompt you to pick a Train Model module. Why? Two ML models are trained in this experiment: one using a Two-Class Boosted Decision Tree algorithm and the other using a Two-Class Logistic Regression algorithm. Recall that the Two-Class Logistic Regression achieved an accuracy of 81.7 percent, while the Two-Class Boosted Decision Tree achieved 80.6 percent. The model I want to expose as a Web service, therefore, is the model created with the Two-Class Logistic Regression algorithm.

The right side of the experiment layout—after the project splits into two paths—appears as what's shown in **Figure 3**. The Tune

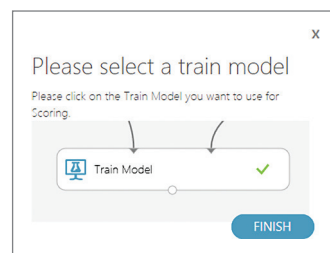


Figure 2 Azure Machine Learning Studio Will Prompt You to Pick a Train Model Module

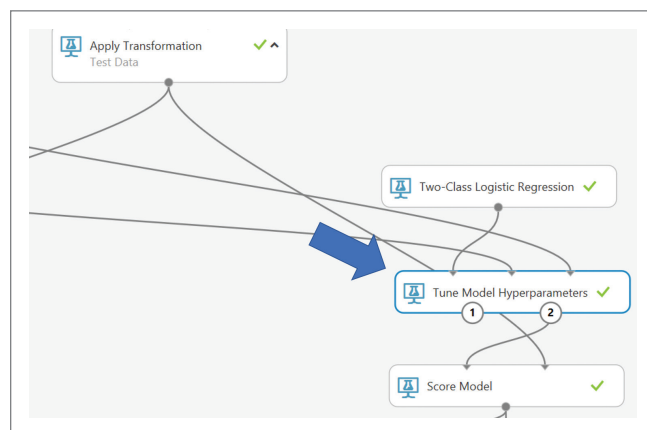


Figure 3 The More Accurate Model Is the Two-Class Logistic Regression, Enhanced by Tune Model Hyperparameters

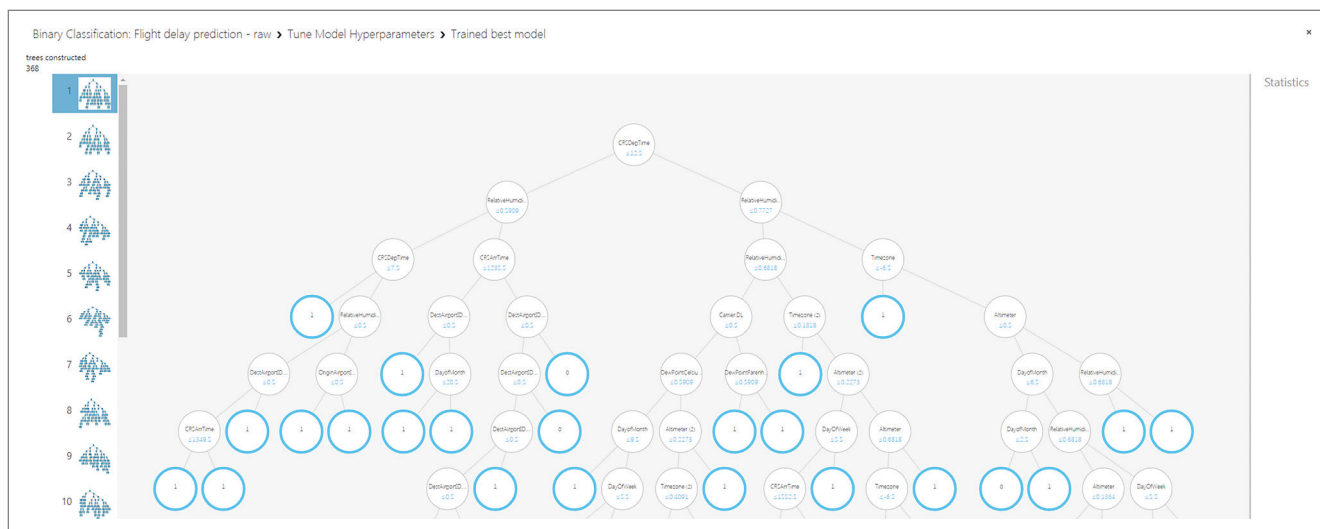


Figure 4 Visualizing a Tune Model Hyperparameters Module

Model Hyperparameters module performs a parameter sweep on a model, in this case the Two-Class Boosted Decision Tree just above the module, to determine the best parameter settings. To get an idea of what this module does, right-click on the right-hand output node and choose Visualize.

Once the dialog loads, it should be similar to **Figure 4**. Machine Learning Studio ran through a number of permutations and created 368 different decision trees using the different input data fields as decision points to find the optimal decision tree. Clicking on any node in a tree will reveal statistics about the node and its impact on creating the output model. The order in which the trees are displayed in the left-hand column indicate their rank in terms of producing an accurate model. Feel free to explore this dialog to get an appreciation of all the work that this module performed.

When you're ready to proceed, close out this dialog. Make sure that the Tune Model Hyperparameters module is selected and then click the Set Up Web Service button. In the context menu that follows, click the Predictive Web Service [Recommended] option. In a few moments, the modules in the experiment will move around and the experiment will split into two tabs: Training Experiment and Predictive Experiment. The work done so far is preserved in the Training Experiment Tab. The Predictive Experiment tab contains the same trained model and data wrangling methods with modifications needed to expose the experiment as a Web service. Click Run to execute the experiment. Once the experiment finishes running, click on the Deploy Web Service button to complete the process of exposing this data model as a Web service.

The Web service dashboard appears on screen once the Web service is deployed. The Web service dashboard provides information on how to access, configure and test the Web service. Click on the Test link highlighted in **Figure 5**.

Click the button to enable Sample Data on the screen that follows. Enabling Sample Data exposes a subset of the training data in the test dialog. The goal is to make the Web service easier for users to begin using the Web service. After a moment, the text fields on the form are populated with values. Click the Test Request-Response button to test the Web service. Test results will appear in the right-hand side of the screen after a few moments.

Consuming the Trained Model in Excel

Now that the trained model has been deployed as a Web service it can be consumed by any HTTP REST-capable client, including Excel 2013 and newer. Excel is an excellent client application for any predictive model, as it's the de facto analytics tool for business users worldwide. Furthermore, Excel integrates with numerous other analytics tools, such as Power BI.

Installing the Azure Machine Learning Add-In

To consume models from Azure Machine Learning you need to install the Azure Machine Learning add-in to Excel. Click on the Insert tab and click the Store button (see **Figure 6**).

In the following dialog click on Store and type "Azure Machine Learning" into the search box and hit Enter. As seen in **Figure 7**, the Azure Machine Learning add-in is the first result. Click Add to install it.



Figure 5 Web Service Dashboard in Azure Machine Learning Studio

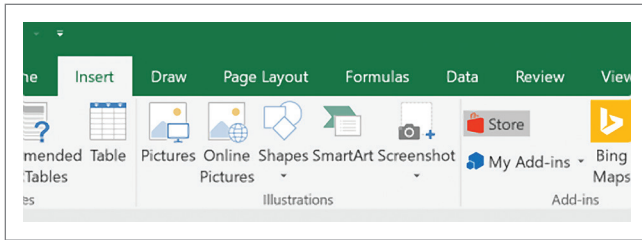


Figure 6 Installing an Add-in to Excel 2016

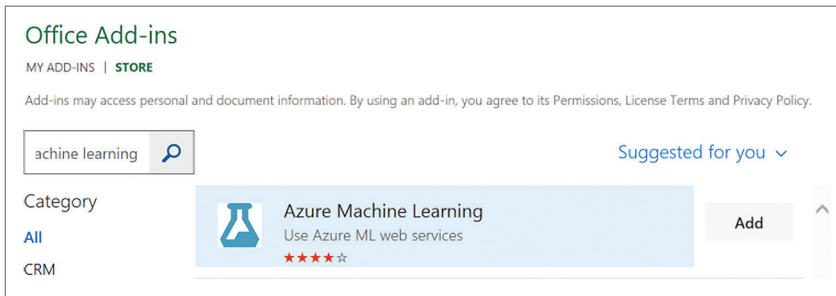


Figure 7 Searching the Store for the Azure Machine Learning Add-in

Once the add-in is installed, an Azure Machine Learning pane appears on the right-hand side of the spreadsheet. Note that there are services already added by default. Click on the Add Web Service button to add the Web service just created. The Add Service dialog in the Azure Machine Learning pane will then display two textboxes: One for the Web service URL and another for the API key to access the service. The API key is found on the Web service dashboard in **Figure 5**. The URL can be found on the same dashboard. In the Default Endpoint section, right-click the text link Request/Response and choose Copy Link Address to copy the URL into the clipboard. Paste that value into the URL textbox in Excel, as shown in **Figure 8**.

Click Add to Add the Web Service to the Current Spreadsheet.

Consuming the Web Service from Excel

Once the Web service is added you can use it. In the Azure Machine Learning pane, click Binary Classification – Flight Delay prediction from the list of available services. Click Use Sample Data and, in a moment, the spreadsheet is populated with a table of sample values. In the Input textbox, enter “Sheet1!A1:N6” to include all of the sample data (assuming that the worksheet is named Sheet1). In the Output textbox enter “O1.” Immediately below the Output text box a warning message will display about overriding existing values. Click Got It to dismiss the warning message.

Click Predict and the predicted values display in a few moments. The Add-in and

link to the Web service are now part of the Excel file. You may now save and share the file with interested parties. Other users won't need to subsequently install the Add-in or configure the Web service, as the add-in installation and configuration has been taken care of for them.

Accessing the Web Service in Other Applications

Navigate to the Web service dashboard displayed in **Figure 5**. Note that—as of this writing—there is a new Web service experience in

preview. Click the link to view the new Web service dashboard experience. Under Basics, click the “Use endpoint” link to view a number of options on how to consume the service.

In addition to the API access keys and service endpoints, there's sample code for accessing the service in C#, Python and R. There's also a link to the Azure ML Request-Response Service Web App project that will instantly create an Azure Website that consumes the data, which can be found at bit.ly/2wBHE1j. For

more information and for a detailed walk-through on how this template works, please refer to the documentation at bit.ly/2yt99Ye.

Wrapping Up

In this article, I demonstrated how to expose ML models created in Machine Learning Studio to users in the outside world. Starting with a predictive model created in the previous column, I demon-

strated exposing that model to more than just data scientists. After all, a predictive model is only as useful as it is accessible to decision makers. Much of the rapid pace of innovation in machine learning lately can be attributed to making Azure Machine Learning technology widely available to business users and consumers.

Leveraging the cloud, these models may be deployed and scaled without any additional infrastructure burden on the organization's existing IT Systems. In fact, little additional effort is required on the part of developers to integrate predictive models from Machine Learning Studio. Now that it's *this* simple to make these models widely available for exploratory analysis in Excel and integration into other applications, the potential to add value with machine learning is limitless. ■

FRANK LA VIGNE leads the Data & Analytics practice at Wintellect and co-hosts the DataDriven podcast. He blogs regularly at FranksWorld.com and you can watch him on his YouTube channel, “Frank's World TV” (FranksWorld.TV).

THANKS to the following technical experts for reviewing this article: [Andy Leonard](#)

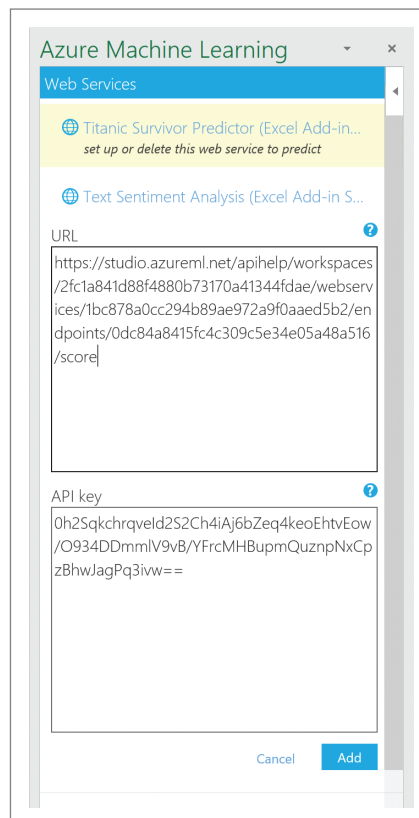


Figure 8 Adding an Azure Machine Learning Web Service to Excel



DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.



Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

All trademarks or registered trademarks are property of their respective owners.



How To Be MEAN: Angular Forms

Welcome back again, MEANers.

All throughout this series, the focus has been about Angular; specifically, using Angular to display data and route through multiple pages. As much as the Web browser is built to be able to display data, however, it's also intended to be able to gather data and pass it back to the server, and thus far, that's been missing from the discussion.

Angular can certainly capture data through forms, but doing so is a touch tricky—not so much in the syntax of creating and defining the browser form, but in one particular aspect of the underlying behavior, which I'll go over later. But let's not put the cart before the horse. Time to talk about some simple form definition and capture.

Formed Components

In previous columns, I've talked about how you can bind “template statements” (to use the Angular terminology) to the events offered up by an Angular component. The most common example of that is capturing the click event on a `<button>` element to invoke a method on an Angular component:

```
<button (click)="console.log('Clicked')">Push me!</button>
```

In and of itself, this is good, but it doesn't provide any facility to capture input—in order for this to be useful for data entry, one of two things has to happen: Either the code in the component has the ability to reference the controls on the page from inside the component (which is what ASP.NET Web Forms, among other frameworks, will do), or the code being invoked has to be able to receive the input data as a parameter. However, this can take a couple of different forms.

First, and most generic, the Angular template statement can reference the `$event` object, which is essentially the DOM's event object generated during the user's session. This simply requires referencing the parameter as part of the statement, such as:

```
<button (click)="capture($event)">Save</button>
```

The drawback, however, is that the object passed is the DOM event representing the user's action; in this case, a `MouseEvent` tracking the location on the screen where the mouse was clicked, and it doesn't really capture the state of the other elements on the page. While it would certainly be possible to navigate the DOM hierarchy to find the control elements and extract the values from them, it's also not really the Angular Way. Components should be isolated away from the DOM, and the template statement should be able to obtain the data it needs and pass it in to a method on the component for use.

This approach suggests that Angular needs some way to identify the input fields on the page, so that the template statement can pull values and pass them in. The need to be able to identify the form element takes the form of an “identifier” on the `<input>` element itself, what Angular calls a “template reference variable.” Like some of the other Angular syntax, it deliberately uses syntax that doesn't look like HTML:

```
<input #firstName>
```

This will create an Input field in HTML, as per the normal HTML tag of the same name, but then introduce a new variable into the template's scope, called `firstName`, which can be referenced from the template statement bound to an event from a field, like so:

```
<button (click)="addSpeaker(firstName, lastName)">Save</button>
```

This is pretty self-explanatory: On a button click, invoke the `addSpeaker` method of the component, passing in the `firstName` and `lastName` variables, accordingly, as shown in **Figure 1**.

However, written like this, what shows up in the browser's console isn't the expected strings from the input; instead, values such as `<input _ngcontent-crf-2>` appear in place of each of those values. The reason for this is simple: The browser console returns the actual Angular representations of the DOM element, rather than the input data that was typed in. The solution to that is equally simple: Make use of the “value” property on each side of the template statement to get to the data the user typed in.

Thus, if I need to build a component for creating new speakers, I can create a component that displays two `<input>` fields, a `<button>` that has a `(click)` that calls `addSpeaker`, passing in `firstName.value` and `lastName.value`, and use that method to invoke the `SpeakerService` (from an earlier article) to save it to the database. But this idea of “creating a new Speaker” is conceptually very close to “editing an existing Speaker,” so much so that some modern databases have

Figure 1 Invoking the addSpeaker Method

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-speaker-edit',
  templateUrl: './speaker-edit.component.html',
  styleUrls: ['./speaker-edit.component.css']
})
export class SpeakerEditComponent implements OnInit {
  constructor() {}
  ngOnInit() {}
  addSpeaker(fname: string, lname: string) {
    console.log("addSpeaker(", fname, ",", lname, ")")
  }
}
```

File Format APIs

Open, Create, Convert, Print and Save files from your applications!

Try risk free - 30 day trial

Aspose.Total

Enable your applications to manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats for all major platforms.



Aspose.Words

Create, edit, convert or print Word documents (DOC, DOCX, RTF etc.) in your .NET, Java and Android applications.



Aspose.Cells

Develop high performance .NET, Java and Android applications to Create, Edit or Convert Excel worksheets (XLS, XLSX, ODS etc).



Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS etc.) using our native APIs for .NET, Java and Android platforms.



Aspose.Slides

Create, edit or convert PowerPoint presentations (PPT, PPTX, ODP etc.) in your .NET, Java and Android applications.



Aspose.Email

Create, Edit or Convert Outlook Email file formats (MSG, PST, EML etc.) and popular network protocols.



Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet etc.) using our native APIs for .NET and Java.

► Aspose.Imaging ► Aspose.Tasks ► Aspose.OCR ► Aspose.Diagram ► Aspose.Note ► Aspose.HTML



ASPOSE
File Format APIs

Download a Free Trial at
<https://downloads.aspose.com>

Americas: +1 903 306 1676

EMEA: +44 141 628 8900
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

Figure 2 The SpeakerEdit Component

```
export class SpeakerEditComponent implements OnInit {  
  
  @Input() speaker: Speaker;  
  @Output() onSave = new EventEmitter<Speaker>();  
  
  constructor(private speakerService: SpeakerService) { }  
  
  ngOnInit() {  
    if (this.speaker === undefined) {  
      this.speaker = new Speaker();  
    }  
  }  
  
  save(fn: string, ln: string) {  
    this.speaker.firstName = fn;  
    this.speaker.lastName = ln;  
    this.onSave.emit(this.speaker);  
  }  
}
```

begun to talk about the insert and update operations as being essentially one and the same: the upsert. It would be quite nice—and component-oriented—if the SpeakerEdit component could serve as either create or edit within the same component.

Thanks to the power of the @Input and @Output directives you've seen earlier, this is actually quite trivial to do. Add an @Input field for the speaker for one to be passed in, and an @Output field to let people know when the user clicks Save. (The latter isn't necessary, if you make the decision that the SpeakerEdit component will always save to the database, and there's no other action that a client of the component would ever want to do. That would be a highly context-sensitive conversation in the team meeting.)

This leaves me with what's in **Figure 2** for the SpeakerEdit component code.

And, as you might expect given my design skills, the template is pretty bare-bones but functional:

```
<div>  
  Speaker Details: <br>  
  FirstName: <input #firstName><br>  
  LastName: <input #lastName><br>  
  <button (click)="save(firstName.value, lastName.value)">Save</button>  
</div>
```

Notice, again, that I use the “.value” to extract the string values out of the firstName and lastName input fields.

Using this component (in this exercise, from the main AppComponent) is pretty straightforward:

```
<h3>Create a new Speaker</h3>  
<app-speaker-edit (onSave)="speakerSvc.save($event)"></app-speaker-edit>
```

In this case, I chose to not save the Speaker from within the component, but have clients do so from the emitted event. The speakerSvc object is a dependency-injected SpeakerService, from the previous article on building services in Angular (msdn.com/magazine/mt826349).

This is what componentization buys you: The ability to create UI “controls” that you can then drop in and just use, rather than have to think about how they work internally.

Firing Events

It would be nice to wrap up here, but there's one gotcha about events in Angular that deserves attention. It's quite common for developers to want to trap things like keystrokes and react to the entered data—for example, to display some auto-complete suggestion values.

The traditional DOM events to this are methods like onBlur or onKeyUp. For example, say it would be nice to track each keystroke and display it as the user is typing. Developers new to Angular might expect this to work:

```
@Component({  
  selector: 'loop-back',  
  template: `  
    <input #box>  
    <p>{{box.value}}</p>  
  `,  
})  
export class LoopbackComponent { }
```

When run, however, this code will not display each character as it's typed—in fact, it'll do nothing at all. This is because Angular won't fire events unless the browser fires an event. For that reason, Angular needs an event to fire, even if the code that gets fired in that event is a complete no-operation, such as the template statement 0, as in:

```
@Component({  
  selector: 'loop-back',  
  template: `  
    <input #box (keyup)="0">  
    <p>{{box.value}}</p>  
  `,  
})  
export class LoopbackComponent { }
```

Notice the “keyup” binding. This tells Angular to register for key events on the Input element, and that gives Angular the opportunity to trigger events, which will then update the view. It's a little awkward to work with at first, but this way Angular isn't polling for any kind of event all the time and, thus, doesn't have to consume quite so many CPU cycles.

Wrapping Up

Some veteran Web developers might find this all a little confusing and awkward: “Why can't we just go back to the good-ol' HTML form?” The Angular Way isn't always obvious, but in many ways, once the reasoning and rationale become clear, they're usually understandable and reasonable. In this particular case, the Angular Way is to embrace the concept of components and think in terms of constructing usable “constructs” that know how to capture input and process it. This permits a degree of flexibility that wasn't really present in the “old Web” way of thinking, such as having multiple such components on a single page. (When you have to refresh the whole page to display results, you end up having one input-validate-save-render cycle per input requirement.)

However, there are some other issues here, such as knowing how to update other parts of the system when a user edits an existing Speaker. Angular has a solution for this, but there are a few more steps to go before being able to get into the world of Reactive Programming. The home stretch is near, though, so stick with me for just a few more. But, as always, in the meantime ... Happy coding! ■

TED NEWARD is a Seattle-based polytechnology consultant, speaker and mentor, currently working as the director of Developer Relations at Smartsheet.com. He's written a ton of articles, authored and coauthored a dozen books, and works all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.

THANKS to the following Microsoft technical expert for reviewing this article:
James Bender

Data Quality Made Easy. Your Data, Your Way.



Melissa provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email, phone, and more.

Our data quality solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools and plugins for the **Microsoft® Product Ecosystem**.



Start Your Free Trial
www.Melissa.com/msft-pd

Melissa Data is Now Melissa.

Why the change?

See for Yourself at the New www.Melissa.com

melissa™

1-800-MELISSA

Code Editing and Debugging in Visual Studio for Mac

Alessandro Del Sole

Visual Studio for Mac is the powerful, new native development environment from Microsoft that perfectly embodies the company's mobile-first, cloud-first vision. It helps you build cross-platform apps with Xamarin and .NET Core—and games with Unity—using your existing .NET skills and your favorite programming languages, such as C# and F#. You'll find an introduction to Visual Studio for Mac at msdn.com/magazine/mt790182. In this article, I'll focus on some powerful features in the code editor and the debugging tools that will help you increase your productivity.

A Roslyn-Powered Code Editor

Visual Studio for Mac allows you to write code in C# and F# on macOS. This is possible because the IDE leverages the .NET Compiler Platform, also known as Project “Roslyn” (github.com/dotnet/roslyn). Roslyn provides open source, cross-platform .NET compilers that

expose rich code analysis APIs. As in Visual Studio 2015 and 2017 on Windows, Roslyn powers the code editor in Visual Studio for Mac, providing an enhanced coding experience with syntax colorization, IntelliSense, live code issue detection, fixes and refactorings. The official documentation explains how to use code fixes and refactorings in general (bit.ly/2jKt69D), so in this article I'll focus more specifically on features and capabilities such as generating types, removing redundant code, navigating code, adding support for custom languages and code snippets. All of these features are available to both Xamarin and .NET Core projects.

Roslyn provides open source, cross-platform .NET compilers that expose rich code analysis APIs.

This article discusses:

- Refactoring redundant code and generating types on-the-fly
- Navigating between code files and code blocks
- Working with built-in and custom code snippets
- Support for custom languages
- Productivity tips for the built-in Debug pads

Technologies discussed:

Visual Studio for Mac

Generating Types On-the-Fly

One of the nicest productivity features in the code editor is the ability to generate new types while coding, without losing your focus on the active editor. For example, you can write the name of a type that doesn't exist yet, and when the code editor highlights the type name as a code issue, you can right-click it (or press Alt+Enter), select Quick Fix | Generate Type. **Figure 1** shows an example based on generating a type called Person.

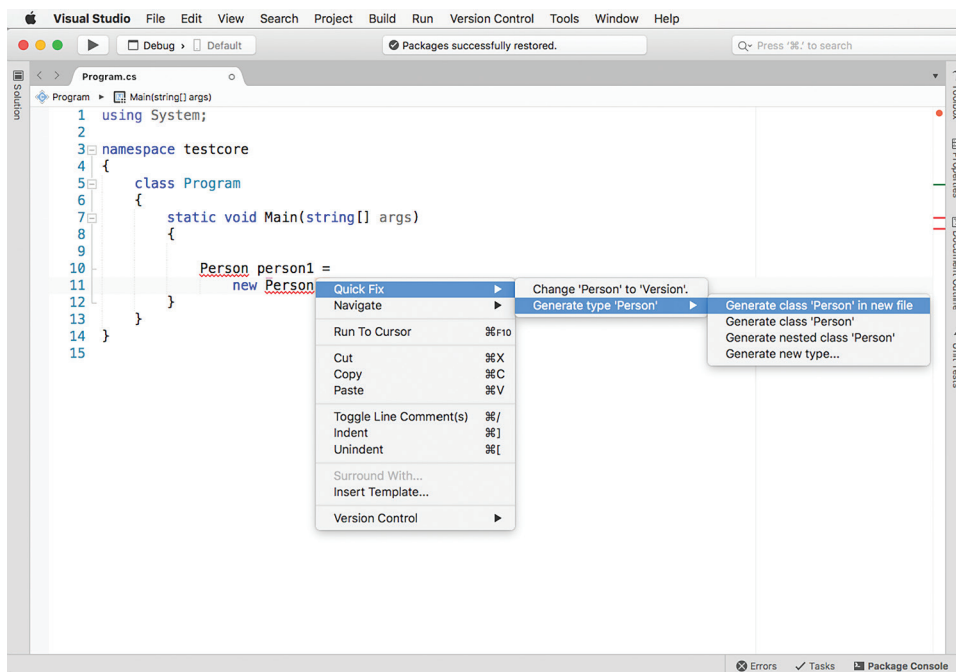


Figure 1 Generating a New Type While in the Active Editor

The first option, Generate class 'Person' in new file, will generate a new class called Person with the internal modifier inside a new file called Person.cs. The second option, Generate class 'Person', will generate a new class called Person with the internal modifier inside the current file. The third option, Generate nested class 'Person', will generate a new private class called Person nested inside the type that's currently active in the code editor (in this case the new class would be generated inside the Program class). You can then change the internal or private modifier either manually or by right-clicking the modifier and then still selecting Quick Fix. In all cases, the new class is generated with an empty constructor. The

code editor will also assign the proper type to the newly generated member, which means it will generate a property (or field) of type string. The code editor can analyze method invocations and generate matching method signatures. This feature is part of the code refactoring tooling and helps you keep your focus on the code while writing.

Removing Redundant Code

The code editor in Visual Studio for Mac also highlights redundant code—code that's not necessary and not utilized. Redundant code is easily recognizable because it's grayed out. Behind the scenes,

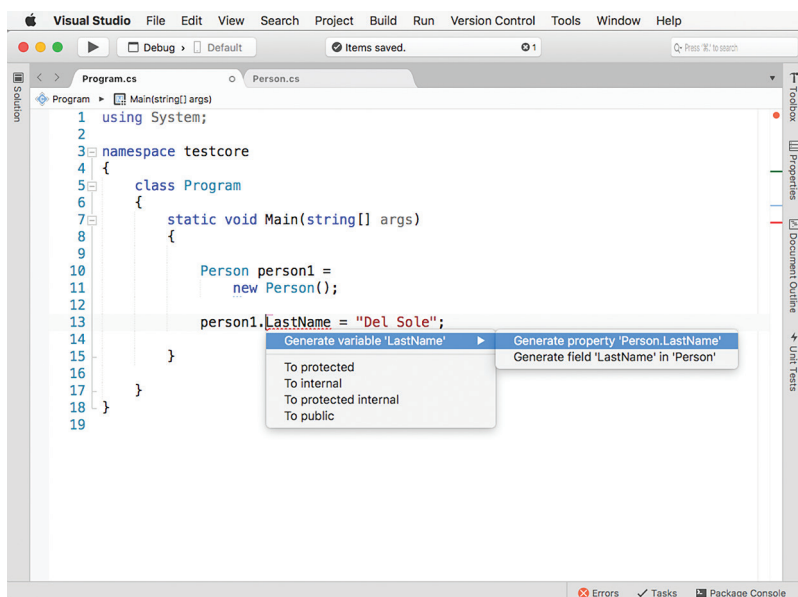


Figure 2 Generating a New Member

redundant code is highlighted based on some source analysis rules that cause the redundant code to be considered an issue. (You can control these rules, but that's out of scope here.) Most of the time you'll find examples of redundant code based on unnecessary using directives, but redundant code recognition isn't limited to this. For example, an empty constructor and the internal modifier are highlighted as redundant in Figure 3.

If you right-click some redundant code, which you recognize by its lighter color, you can then select Quick Fix and Visual Studio will show a code fix that will remove the unnecessary code. Additionally, you'll see a submenu called Options that allows you to:

- Suppress the current issue detection.
- Configure the analysis rule further in the Visual Studio preferences.
- Apply the code fix for multiple issues of the same type at the document, project or solution level.

Similarly, in the earlier example you can easily fix redundant internal modifiers for type definitions (internal is, in fact, the default modifier in C#). This technique applies to any analysis rule that highlights redundant code in the editor.

Navigating Code

Navigating between code files and between members in a code file is extremely common and having built-in, powerful navigation tools significantly aids productivity. Visual Studio for Mac provides a number of robust tools that make it easier to navigate between files, types and members. First of all, you can quickly move between code files by pressing Control+Tab. This action shows a popup where you can see a list of files in your solution. Holding Control and pressing Tab again cycles down the list and then, when you release, the selected file will be opened in the editor. For source code navigation, the next subsections talk about less-known productivity features that are incredibly useful.

The Find All References and Navigate Tools

The Find All References tool allows you to retrieve all the references to a type or member in the solution. To see this tool in action, simply right-click a type or member name in the code editor and then select Find All References. References are shown in the Search Results pad, as shown in Figure 4.

In the Search Results pad, you can see (starting from left to right) the project that contains the reference, the code file including the position, the source text that contained the referenced object, and the full pathname of the source file. You can double-click a reference and the code editor will open the appropriate code file and place the cursor on the selected occurrence. Notice how the source code in the Text column has basic syntax colorization. Find All References is very powerful, but sometimes you might want to filter your search based on certain type and member characteristics. To accomplish this, you can use the Navigate tool, which you invoke by right-clicking a type or member and then selecting Navigate. You'll be presented with a submenu that shows the following search options:

Find References of All Overloads finds all references of a method and its overloads.

Base Symbols allows you to find the list of base types and interfaces that the type on which you invoked Navigate is inheriting from or is implementing. In the case of methods, Navigate will find the list of methods that the current method is overriding.

Derived Symbols allows you to find the list of types that inherit from the type on which you invoked Navigate. In the case of methods, it finds the list of methods that are overriding the one on which you invoked Navigate.

Extension Methods finds all the extension methods for the type on which you invoked Navigate and that are defined in the current solution.

Member Overloads is similar to Extension Methods, but it finds the list of method overloads defined in the current solution.

Implementing Members, if invoked on an abstract class or interface, shows the list of types and members that implement that type or interface.

Visual Studio for Mac
provides a number of robust
tools that make it easier to
navigate between files, types
and members.

The Navigate tool shows the search results in the Search Results pad exactly like Find All References.

The Scrollbar and Minimap Mode

The code editor's scrollbar displays colored markers that represent code issues, such as warnings and errors, breakpoints, ToDo items, and a colored dot at the top that's red if the active file contains errors, yellow if the active file contains warnings, or green if no issues are

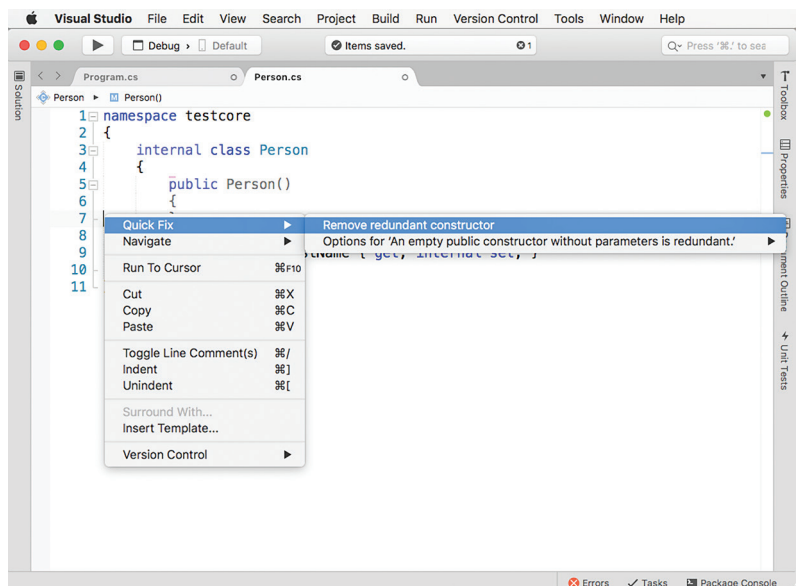


Figure 3 Removing Redundant Code



Figure 4 Finding a Type or Member References

detected. In addition, the scrollbar provides the so-called Minimap mode. When this mode is enabled, the scrollbar displays a preview of the source code for easy navigation, as shown in **Figure 5**.

You enable Minimap mode by right-clicking the scrollbar and selecting Show Minimap. You can click an area on the map and the code editor will move the cursor to the appropriate point. To disable Minimap mode, you right-click the scrollbar again and select Show Tasks. Minimap mode is particularly useful with long files and helps you have a visual representation of the whole file.

Browsing Objects in a Code File

Visual Studio for Mac offers visual ways to easily browse type and members within a code file. Each code editor window shows breadcrumbs that you can click to see a list of the types defined in

the active editor; when you select a type, an additional tab allows you to display the list of its members. Additionally, you can use the Document Outline pad to get a visual representation of the type structure in the active file (see **Figure 6**).

Minimap mode is particularly useful with long files and helps you have a visual representation of the whole file.

Notice how different icons represent the different kinds of members. You might already know the Document Outline pad for its capabilities to show the visual hierarchy of the UI, but you can also use it to get a view of a type's structure, which is very helpful. You can simply double-click an object within Document Outline and the editor will move the cursor to its definition.

Working with Code Snippets

Visual Studio for Mac supports IntelliSense code snippets—pre-written code block templates that can be customized to match your needs. If you have experience with Visual Studio on Windows, you already know what code snippets are. In Visual Studio for Mac, you have two options to insert a code snippet. The first option is right-clicking the code editor and then selecting Insert Template. A list of available code snippets will appear and you just select the one you need. The second option is picking up a code snippet from the IntelliSense completion list as you type. **Figure 7** shows an example where a code snippet is highlighted (you press Tab twice to insert the snippet).

Code snippets are represented with the (...) icon, which makes them immediately recognizable. In both cases, a tooltip describes the code snippet's purpose when you hover over its name with the mouse. Visual Studio for Mac also lets you create new custom code snippets and edit existing ones from within the IDE, without the need of external tools. To accomplish this, select Preferences in the Visual Studio menu, then in the Preferences dialog locate and select the Code Snippets item under Text Editor. There you'll see a list of code snippets grouped by language. If you select an existing snippet, you just press the Edit button to edit the code. If you instead click Add, you will have the option to create a new code snippet. This is done in the New template dialog, where you provide a keyboard shortcut, a description, a MIME type, a language group and, of course, the source code. **Figure 8** shows an example.

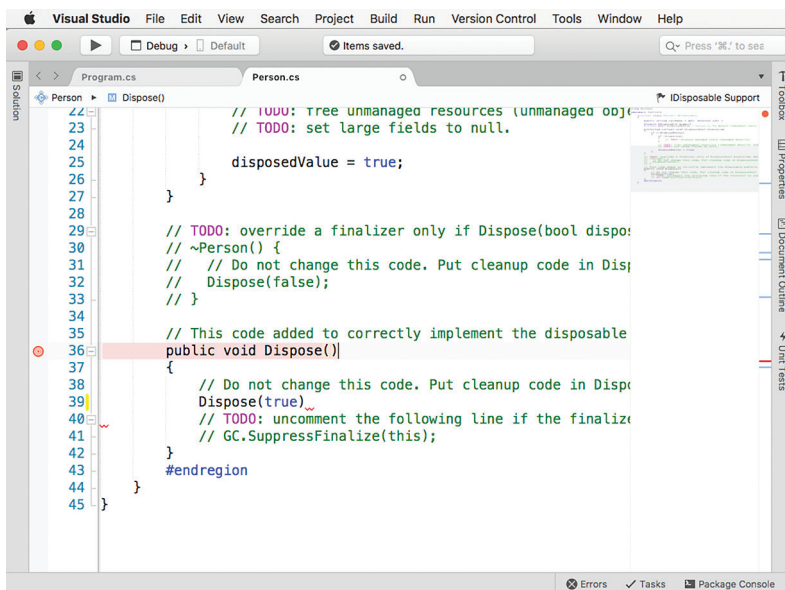


Figure 5 The Scrollbar Minimap Mode

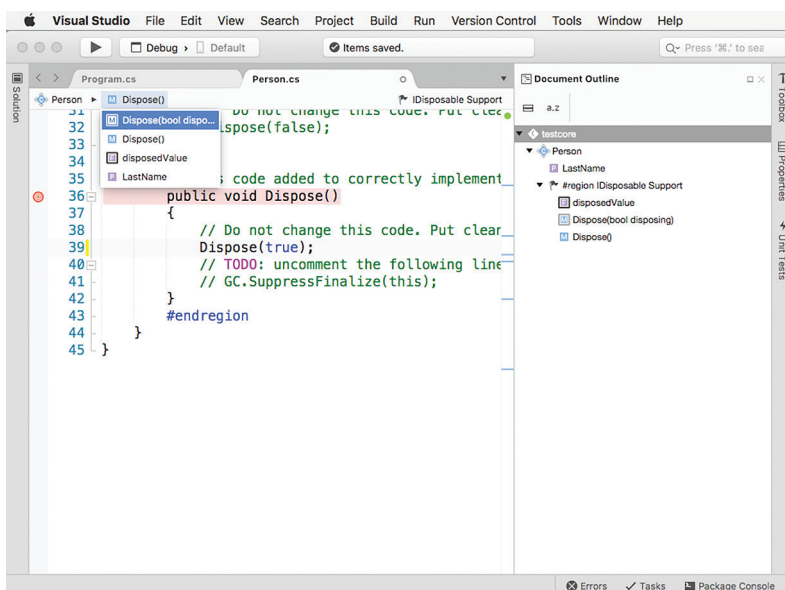


Figure 6 Browsing Objects in a Code File

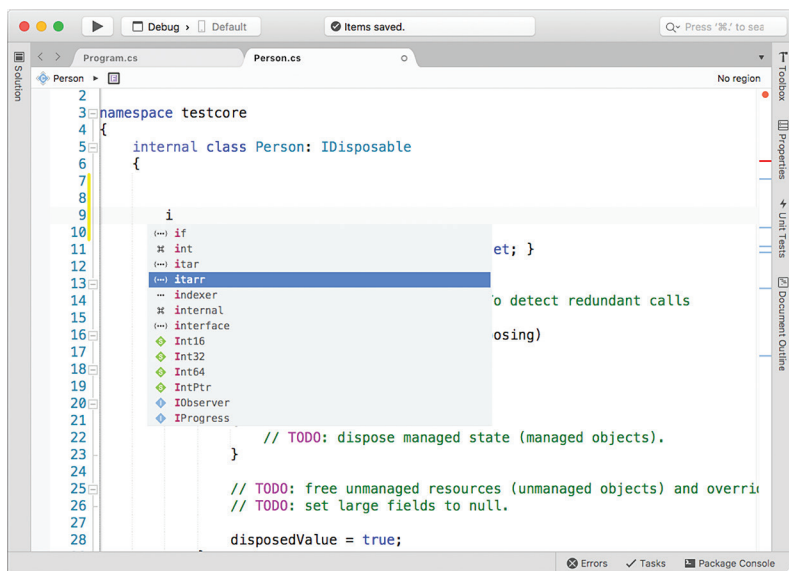


Figure 7 Adding a Code Snippet

Notice that the \$ symbol is used to mark identifiers for replacement, whereas the \$selected\$end\$ expression delimits the current snippet. When you mark identifiers for replacement, you can also provide additional information on the identifier itself, such as its default value and a tooltip that describes its meaning, in the boxes on the right side of the dialog. When you're done, simply click OK and close the Preferences dialog. At this point, your new code snippet is in the snippet library and ready to be used in the code editor through IntelliSense. It's worth noting that you can edit existing snippets to assign a keyboard shortcut if one's not already available. This allows you to type the keyboard shortcut within the code editor and insert a snippet faster. If you're like me and use code snippets a lot, having the option to create them from within the IDE will save you a huge amount of time.

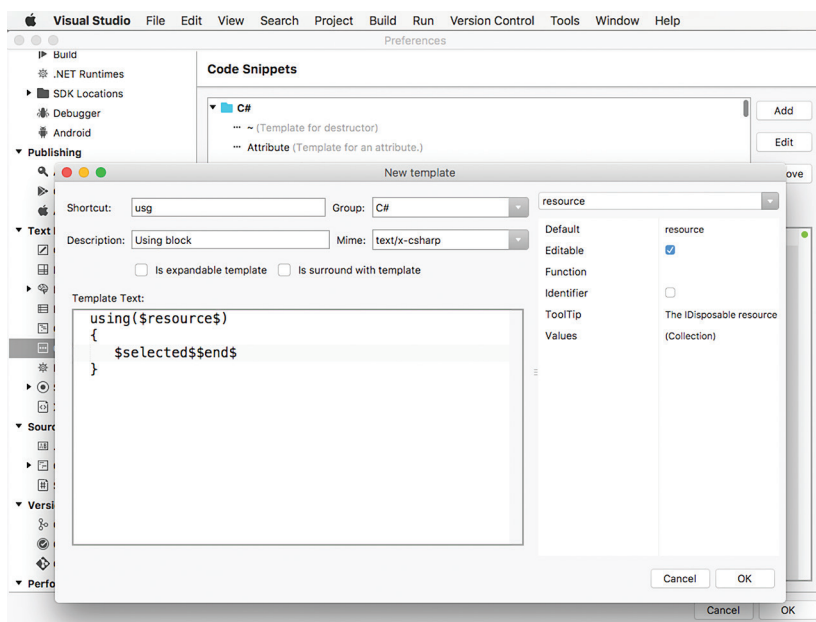


Figure 8 Creating a Custom Code Snippet

Adding Custom Languages

One of the things you'll love in Visual Studio for Mac is the ability to add new languages that aren't included out of the box, whose grammar is based on the TextMate and Sublime Text de facto standards. In fact, Visual Studio for Mac supports both standards and allows adding language bundles offering editing features such as syntax colorization, code snippets and word completion. For example, suppose you want to add syntax support for editing Swift files, which could be very useful on a Mac. In either TextMate or Sublime Text, you install a Swift language bundle and then export the package of the language bundle to disk. Then you can import the language bundle into Visual Studio for Mac. To accomplish this, you use the Language Bundles node of the Preferences dialog. Here you'll be able to click Add and select the language package exported before. At this point, you'll be able to open .swift files (or other file types depending on the bundle you imported) and to take advantage of features such as syntax colorization and code blocks folding.

One of the things you'll love in Visual Studio for Mac is the ability to add new languages that aren't included out of the box.

The code editor also lets you insert code snippets if the language bundle you selected contains any. Obviously, Visual Studio for Mac doesn't support compiling source files or building and publishing applications based on external language bundles. What you can do instead is automate the execution of external tools, such as compilers, with the Edit Custom Tools command in the Options menu.

Debugging Productivity with Pads and the Debug Class

Debugging is tremendously important and Visual Studio for Mac ships with the kind of first-class debugging tools every developer needs to build high-quality applications in a productive way. The official documentation (bit.ly/2xglkx0) describes the most commonly used tools, such as breakpoints, data visualizers and conditional flow control. I'm going to describe some other nice features you might want to know about, especially if you're familiar with debugging in Visual Studio on Windows and you expect to see the same tools on the Mac. Let's start with the System.Diagnostics.Debug class, which lets you print the evaluation of an expression to the Application

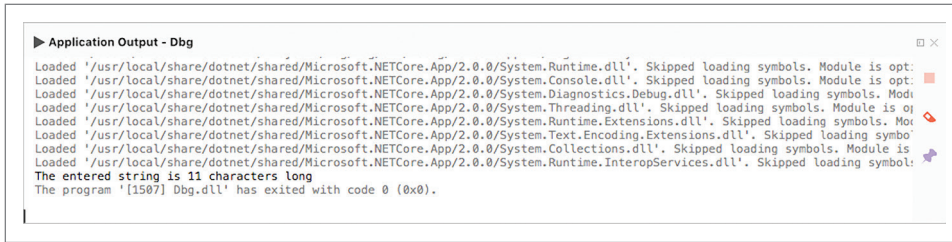


Figure 9 The Result of an Evaluation Printed to the Application Output Pad

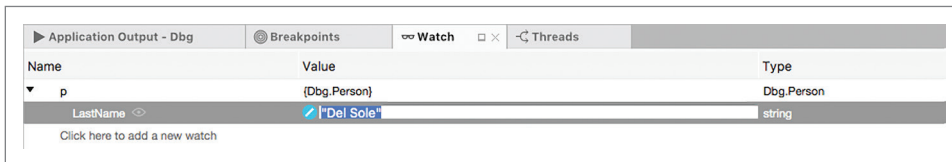


Figure 10 Monitoring Objects with the Watch Pad

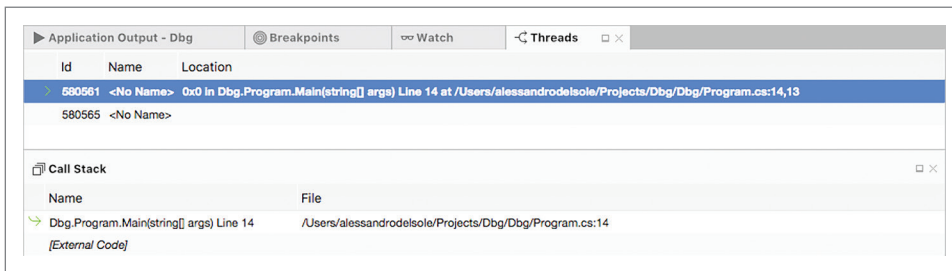


Figure 11 Monitoring Threads and Method Calls with the Threads and Call Stack Pads

Output pad from C# code, without breaking the execution of an application. Suppose you have a .NET Core console application that waits for the user input with the following code:

```
string inputString = Console.ReadLine();
Debug.WriteLine($"The entered string is {inputString.Length} characters long");
```

The `Debug.WriteLine` method prints the expression to the Application Output pad, as depicted in **Figure 9**, without stopping application execution. In this case, the expression is an interpolated string that contains the length of the string entered by the user.

Debugging is tremendously important and Visual Studio for Mac ships with the kind of first-class debugging tools every developer needs to build high-quality applications in a productive way.

And you're not limited to the `WriteLine` method; in fact, you can use all the other supported methods, such as `Assert`, `Equals`, `Fail`, `Write`, `WriteIf` and `WriteLineIf` (see bit.ly/2ydS8j0 for details).

In Visual Studio for Mac, there are other ways to evaluate expressions and to inspect object values while debugging. You can use breakpoints and data visualizers, but you can also use the Watch

pad, whose purpose is to provide a way to visually monitor variables, methods and expressions. While in break mode, the Watch window is automatically enabled and you can click inside it to add a new object to monitor. **Figure 10** shows an example based on a property's value.

For each object you monitor, you can see its members and their values. You can also click the value and change it, so you can see how your code behaves with a different object value. The Watch pad still provides shortcuts to the data visualizers, which you can recognize through the eye and pencil icons. You'll also find other two debugging pads very useful: the Threads pad and the Call Stack pad, both visible in **Figure 11**.

The Threads pad shows the list of running threads and is useful for understanding the code locations in which your app's various threads are paused. It shows the

thread ID, the name and the location of the code that refers to each thread. You can also enter a name for threads if one doesn't exist already. The Call Stack pad shows the hierarchy of method calls, and you can enable it to also display calls to code that's not in your solution; for example, interop frames. To accomplish this, you right-click the pad and enable the Show External Code option. By leveraging all of these, you have a complete and powerful suite of debugging tools that, together with breakpoints, the Locals pad, and data visualizer, give you deep control over your .NET Core and Xamarin solutions.

Wrapping Up

Visual Studio for Mac not only helps you build cross-platform applications for mobile devices and the cloud with Xamarin and .NET Core on macOS, it also offers all the productivity tools you need to write high-quality code. This includes productivity features in the code editor and in the debugging tools that will make you feel at home with this IDE, especially if you have experience with Visual Studio on Windows. ■

ALESSANDRO DEL SOLE has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole works as a senior .NET developer, focusing on .NET and mobile app development, training and consulting. He has recently authored an upcoming book called "Beginning Visual Studio for Mac" (bit.ly/2hsRxYx). You can follow him on Twitter: @progalex.

THANKS to the following Microsoft technical expert for reviewing this article: Mikayla Hutchinson

From Text to Targeted Sentiment Analysis with Cognitive Services

Ashish Sahu

Human languages have evolved over many centuries and, as a result, tend to have complex rules to govern emotional expression. As practitioners of human languages, we typically rely on learning and various social conventions to master one or more languages and use them to express ourselves in different situations. The qualities and learnings are reflected in the responses we leave in various settings, including product reviews, survey responses in online and offline format, and the like. They follow the same language rules and provide a fascinating opportunity to test and advance the machine learning models that work on human language expression.

This article discusses:

- Mining information from customer feedback using Microsoft Cognitive Services tools
- Using the Text Analytics API to find the language, key phrases and sentiment expressed in the text
- Using the Linguistic Analysis API to understand language concepts, structure and insights programmatically

Technologies discussed:

Microsoft Cognitive Services, Text Analytics API, Linguistic Analysis API

Code download available at:

bit.ly/2ic3ykZ

Today, social networks play a great role in shaping popular opinion. We participate in discussions on just about everything, express our thoughts and agree or disagree with each other for all of the world to see. These experiences also spill into our shopping habits and often we leave our views on things we buy and services we try in the form of reviews.

Because we can't physically see and assess the things we want to buy online, we're often left to others' opinions to evaluate how durable or reliable they might be. These reviews may come with star ratings that offer a minimal convenience for sorting through them. However, the star rating doesn't really provide an accurate picture of someone's experience with a product or the quality of the product—sometimes they're just unhappy that an item shipped later than expected.

That leaves us with just one option—to read through the existing reviews and make an educated guess about what's good and bad about the product in question. While that helps, it's time-consuming and inefficient. Now, however, the Text Analytics and the Linguistic Analysis APIs, part of Microsoft Cognitive Services, can help.

Just like the other Cognitive Services, the Text Analytics and Linguistic Analysis APIs are hosted services. There's no software you need to download, configure or train before you start using these artificial intelligence (AI)-based APIs to do amazing things.

The Text Analytics API can infer insights, such as the language of the text, key phrases being discussed and the sentiment expressed in the text. The Linguistic Analysis API, in contrast, enables the

understanding of linguistic concepts and actions in free-form text. It performs part-of-speech (POS) tagging, tokenization and sentence separation, among other things. You can use this API to mine information from customer feedback and to understand the intent of users and the structure of the text.

In this article, I'll show how these APIs work and the scenarios where they can be used to generate insights from the various forms of text encountered every day.

What Does Text Analytics Do?

The Text Analytics API is a suite of services built with Azure Machine Learning. At the time of this writing, you can use the following API operations to perform the actions mentioned earlier:

- *Detect language* determines the language of the text sent to the API from among the 120 supported languages.
- *Key phrases* finds key phrases in the text input.
- *Sentiment* detects the sentiment of the text, returning a numeric score between 0 and 1, where a higher score denotes the degree of positive sentiment.
- *Detect topics* discovers topics or phrases from a corpus of text in a batch operation. This operation is marked for deprecation and won't be covered in the article.

The Text Analytics API is pre-trained to perform these operations and doesn't need any training data. It can handle most of the reviews on most shopping portals. To help you better understand the capabilities of the Text Analytics API, take a look at the following example review:

"This phone has a great battery. The display is sharp and bright. But the store does not have the apps I need."

Figure 1 The Languages Result

```
{
  "documents": [
    {
      "id": "1",
      "detectedLanguages": [
        {
          "name": "English",
          "iso6391Name": "en",
          "score": 1
        }
      ]
    }
  ],
  "errors": []
}
```

Figure 2 The Key Phrases Result

```
{
  "documents": [
    {
      "keyPhrases": [
        "phone",
        "great battery",
        "display",
        "store",
        "apps"
      ],
      "id": "1"
    }
  ],
  "errors": []
}
```

This excerpt reflects a shopper's experience with a phone. Let's run this example through the different API operations and see what turns up.

Along the way, you can use any utility you choose (such as Fiddler, Postman or Curl) for these tests to see the results for yourself. You can also use the API Testing Console, available in the API reference documentation. But there are a few things you need to know before you can test run these APIs:

1. You need a subscription key. If you don't have an Azure subscription, you can get a trial key from bit.ly/2eLG80T.
2. The API expects to receive the text record in the following format:

```
{
  "documents": [
    {
      "language": "string",
      "id": "string",
      "text": "string"
    }
  ]
}
```

This format applies to the key phrases and the sentiment operations. However, you'll need to remove the "language" field for the calls to the detect language operation.

I'll start with a simple test to determine the language of the example text, and the request body will be formatted like so:

```
{
  "documents": [
    {
      "id": "1",
      "text": "This phone has a great battery. The display is sharp and bright. But the store does not have the apps I need."
    }
  ]
}
```

I'm using the Curl utility and here's what my request looks like:

```
curl -v -X POST "https://westus.api.cognitive.microsoft.com/text/
analytics/v2.0/languages"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: my-trial-key"

--data-ascii '{
  "documents": [
    {
      "id": "1",
      "text": "This phone has a great battery. The display is sharp and bright. But the store does not have the apps I need."
    }
  ]
}'
```

And the result I get back after I place this request is shown in **Figure 1**.

As you can see, the result is a scored response and English was detected as the language of the text record submitted. You are welcome to try out a different language to see how it works.

Next, I'll run the same record through the key phrase endpoint. I need to change my Curl command with the updated endpoint and the request body as follows:

```
curl -v --silent -X POST "https://westus.api.cognitive.microsoft.com/text/
analytics/v2.0/keyPhrases"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: my-trial-key"

--data-ascii '{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "This phone has a great battery. The display is sharp and bright. But the store does not have the apps I need."
    }
  ]
}'
```

The response is shown in **Figure 2**.

As you can see, the key phrases endpoint has detected appropriate phrases from the example review.

Now let's see what this review denotes on the scale of positivity. I'll run the same text through the sentiment endpoint and see what comes back. Here's my Curl command:

```
curl -v --silent -X POST "https://westus.api.cognitive.microsoft.com/text/
analytics/v2.0/sentiment"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: my-trial-key"
--data-ascii '{
  "documents": [
    { "language": "en",
      "id": "1",
      "text": "This phone has a great battery. The display is sharp and bright.
        But the store does not have the apps I need."
    }
  ]
}'
```

And the result this time is:

```
{
  "documents": [
    {
      "score": 0.770478801630976,
      "id": "1"
    }
  ],
  "errors": []
}
```

The outcome is simpler this time, and you can see that the sentiment score for the review is .77, which, on a scale of 0 to 1 is 77 percent. This denotes a mostly positive sentiment for the text—which you can infer from text.

Now that I've run this text review with all the available operations, I'll combine them to see the overall result:

- Text: This phone has a great battery. The display is sharp and bright. But the store does not have the apps I need.
- Language: English
- Key Phrases: phone, great battery, display, store, apps
- Sentiment: 77 percent

Figure 3 The Result of a Simple GET Request to the List Analyzer Endpoint

```
[
  {
    "id": "4fa79af1-f22c-408d-98bb-b7d7aeef7f04",
    "languages": [
      "en"
    ],
    "kind": "POS_Tags",
    "specification": "PennTreebank3",
    "implementation": "cmm"
  },
  {
    "id": "22a6b758-420f-4745-8a3c-46835a67c0d2",
    "languages": [
      "en"
    ],
    "kind": "Constituency_Tree",
    "specification": "PennTreebank3",
    "implementation": "SplitMerge"
  },
  {
    "id": "08ea174b-bfdb-4e64-987e-602f85da7f72",
    "languages": [
      "en"
    ],
    "kind": "Tokens",
    "specification": "PennTreebank3",
    "implementation": "regexes"
  }
]
```

This experiment demonstrates what the Text Analytics Cognitive Service can do for text reviews, survey responses and customer input. However, combined with the Linguistic Analysis services, I can distil even deeper insights.

A Look at Linguistic Analysis: What's the Difference?

The Text Analytics API uses a pre-trained model to analyze text to detect information such as language, key phrases and sentiments. In contrast, the Linguistic Analysis API uses advanced linguistic analysis tools to process the text inputs and allows you to understand the structure of the sentence. This understanding can then be used to mine information from text records, interpret user commands and process free-form text from any source.

I'm not a linguist, so I'll leave the job of explaining the ins and outs of this service to someone who is. In this article, I'll just cover the basics of this service and then come back to the original intent of this article, which is to generate deeper and meaningful insights from text inputs.

The Linguistic Analysis API uses advanced linguistic analysis tools to process the text inputs and allows you to understand the structure of the sentence.

The Linguistic Analysis API uses the concept of analysers to understand the structure of text records. Currently, three kinds are supported:

- Tokens
- POS Tags
- Constituency Tree

These analyzers come from the Penn Treebank, which is the annotated, parsed text corpus that allows the Linguistic Analysis API to understand whether a given word in a text input is a noun or a verb. For example, "I love Bing!" and "Let me bing this for you" use the word Bing in different capacity.

Let's use this example to understand how Linguistic Analysis works its magic. Just as with Text Analytics, you'll need a trial key if you don't have an Azure subscription.

Once you have the key, just fire up your tool of choice to send a request to the Linguistic Analysis API. There are two operations available with this API:

- *List analyzers* return the list of analysers to parse the text for Tokens, POS Tags and a Constituency Tree.
- *Analyze text* parses the text inputs you provide using the analyzers you supply in the request.

Figure 3 shows what a simple GET request to the List Analyzer endpoint returns.

I'll use the previously mentioned analyzers to parse this text: "I Love Bing! Let me Bing this for you," formatting the request body as follows:



Help & Manual Professional | from \$586.04



Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePub, Kindle or print
- Styles and Templates give you full design control

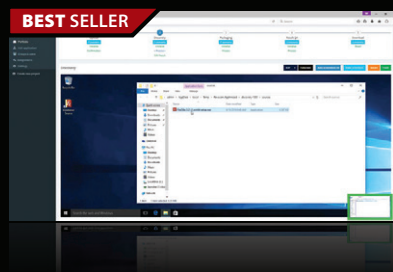


DevExpress DXperience 17.1 | from \$1,439.99



The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics

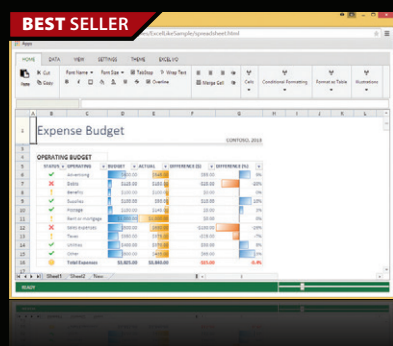


Apptimized | from \$5,292.00



Discover, package, test and manage applications in the cloud.

- Rapidly test and prepare apps for Windows 10 without the need for hardware or software
- Capture a default install automatically and create customized discovery documentation
- The service is configured on a tenant philosophy, meaning that customer data is protected
- The advanced MSI Editor provides full MSI and App-V editing capability
- Supports bulk upload source media, and zip file capability for requests that have multiple EXEs



SpreadJS | from \$984.02



Deliver intuitive, efficient, multi-functional, pure JavaScript spreadsheets for Enterprise apps.

- Harness the power of a spreadsheet to display and manage data like Microsoft Excel
- Go beyond the grid with cards, trellis, calendar, Gantt, news feed, timeline and more
- Renders to the HTML canvas for a fast, interactive user experience across all browsers
- Modularized - so you only need to load the JavaScript that contains the features you need
- A Client-side component - works with Windows, Linux, MacOS, Android and iOS

```
{
  "language" : "en",
  "analyzerIds" : ["4fa79af1-f22c-408d-98bb-b7d7aeef7f04",
    "22a6b758-420f-4745-8a3c-46835a67c0d2",
    "08ea174b-bfdb-4e64-987e-602f85da7f72"],
  "text" : "I love Bing! Let me bing this for you"
}
```

Here's my Curl command:

```
curl -v --silent -X POST https://westus.api.cognitive.microsoft.com/
  linguistics/v1.0/analyze
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: my-trial-key"
--data-ascii '{
  "language" : "en",
  "analyzerIds" : ["4fa79af1-f22c-408d-98bb-b7d7aeef7f04",
    "22a6b758-420f-4745-8a3c-46835a67c0d2",
    "08ea174b-bfdb-4e64-987e-602f85da7f72"],
  "text" : "I love Bing! Let me bing this for you"
}'
```

The response to this request is shown in **Figure 4**.

The result is segregated based on each of the analyzers sent in the request—POS Tags, Constituency Tree and Tokens in this example.

As you can see, the POS Tags are just tags for each of the words in the text input, while the Constituency Tree analyzer returns the tree structure of the text input marked with tags and the words. The Tokens analyzer returns the most readable result where it includes the information about each of the words in the text input along with their position in the record.

For this article, I'll be using the Constituency Tree and Tokens analyzers to break text reviews into separate records based on Sentence Separation information and the Conjunction words.

If you'd like to read more about the Linguistic Analysis API and related concepts, I encourage you to read the complete API documentation available at bit.ly/2eTc2Nj.

Now let's go back the original example used for the Text Analytics API: "This phone has a great battery. The display is sharp and bright but the store does not have the apps I need."

There's a subtle change in the example this time as I've removed the period at the end of the second sentence of the original example

Figure 4 Parsing the Example Text

```
[
  {
    "analyzerId": "4fa79af1-f22c-408d-98bb-b7d7aeef7f04",
    "result": [
      [
        "PRP",
        "VBP",
        "VBG",
        "."
      ],
      [
        "VB",
        "PRP",
        "JJ",
        "DT",
        "IN",
        "PRP"
      ]
    ]
  },
  {
    "analyzerId": "22a6b758-420f-4745-8a3c-46835a67c0d2",
    "result": [
      "(TOP (S (NP (PRP I)) (VP (VBP love) (NNP Bing)) (. !)))",
      "(VP (VBD Let) (S (NP (PRP me)) (VP (VBG bing) (NP (DT this)) (PP (IN for) (NP (PRP you))))))"
    ]
  },
  {
    "analyzerId": "08ea174b-bfdb-4e64-987e-602f85da7f72",
    "result": [
      {
        "Len": 12,
        "Offset": 0,
        "Tokens": [
          {
            "Len": 1,
            "NormalizedToken": "I",
            "Offset": 0,
            "RawToken": "I"
          },
          {
            "Len": 4,
            "NormalizedToken": "love",
            "Offset": 2,
            "RawToken": "love"
          },
          {
            "Len": 4,
            "NormalizedToken": "Bing",
            "Offset": 7,
            "RawToken": "Bing"
          }
        ]
      },
      {
        "Len": 1,
        "NormalizedToken": "!",
        "Offset": 11,
        "RawToken": "!"
      }
    ]
  },
  {
    "Len": 24,
    "Offset": 13,
    "Tokens": [
      {
        "Len": 3,
        "NormalizedToken": "Let",
        "Offset": 13,
        "RawToken": "Let"
      },
      {
        "Len": 2,
        "NormalizedToken": "me",
        "Offset": 17,
        "RawToken": "me"
      },
      {
        "Len": 4,
        "NormalizedToken": "bing",
        "Offset": 20,
        "RawToken": "bing"
      },
      {
        "Len": 4,
        "NormalizedToken": "this",
        "Offset": 25,
        "RawToken": "this"
      },
      {
        "Len": 3,
        "NormalizedToken": "for",
        "Offset": 30,
        "RawToken": "for"
      },
      {
        "Len": 3,
        "NormalizedToken": "you",
        "Offset": 34,
        "RawToken": "you"
      }
    ]
  }
]
```

and used the conjunction word “but” to combine it with the third sentence. You’ll see that I have a purpose in doing so.

I’ll now call the Analyze Text operation of the Linguistic Analysis API with the following request body:

```
{
  "language": "en",
  "analyzerIds": ["22a6b758-420f-4745-8a3c-46835a67c0d2", "08ea174b-bfdb-4e64-987e-602f85da7f72"],
  "text": "This phone has a great battery.
  The display is sharp and bright but the store does not have the apps I need."
}
```

I’m sending this request with the Constituency Tree and Tokens analyzers only. **Figure 5** shows the result.

As you can see, the Constituency Tree analyzer broke the text record in to two sentences and indicated the conjunction word “but” with the tag CC in the result. As noted, I purposely removed the

period character at the end of the second sentence in the original text in order to highlight this capability of the Linguistic Analysis API.

Better Together

This little experiment demonstrates an extremely reliable way to break the original example into three separate sentences, which can now be processed using the Text Analytics API to gain deeper insight about the customer input. You might think that simple string parsing and splitting would provide a similar result offline, but that’s easier said than done—take my word for that.

Now, in the same manner, I can take the output of the Linguistic Analysis API and, with a simple string-manipulation routine, in any programming language, separate the review into individual parts. The original example can be broken down into the following three parts:

Figure 5 Using the Analyze Text Operation of the Linguistic Analysis API

<pre>[{ "analyzerId": "22a6b758-420f-4745-8a3c-46835a67c0d2", "result": ["(TOP (S (NP (DT This) (NN phone)) (VP (VBZ has) (NP (DT a) (JJ great) (NN battery))) (. .)))", "(TOP (S (S (NP (DT The) (NN display)) (VP (VBZ is) (ADJP (JJ sharp) (CC and) (JJ bright)))) (CC but) (S (NP (DT the) (NN store)) (VP (VBZ does) (RB not) (VP (VB have) (NP (NP (DT the) (NNS apps)) (SBAR (S (NP (PRP I)) (VP (VBP need)))))) (. .)))"] }, { "analyzerId": "08ea174b-bfdb-4e64-987e-602f85da7f72", "result": [{ "Len": 31, "Offset": 0, "Tokens": [{ "Len": 4, "NormalizedToken": "This", "Offset": 0, "RawToken": "This" }, { "Len": 5, "NormalizedToken": "phone", "Offset": 5, "RawToken": "phone" }, { "Len": 3, "NormalizedToken": "has", "Offset": 11, "RawToken": "has" }, { "Len": 1, "NormalizedToken": "a", "Offset": 15, "RawToken": "a" }, { "Len": 5, "NormalizedToken": "great", "Offset": 17, "RawToken": "great" }, { "Len": 7, "NormalizedToken": "battery", "Offset": 23, "RawToken": "battery" }, { "Len": 1, </pre>	<pre> "NormalizedToken": ".", "Offset": 30, "RawToken": "."] }, { "Len": 76, "Offset": 32, "Tokens": [{ "Len": 3, "NormalizedToken": "The", "Offset": 32, "RawToken": "The" }, { "Len": 7, "NormalizedToken": "display", "Offset": 36, "RawToken": "display" }, { "Len": 2, "NormalizedToken": "is", "Offset": 44, "RawToken": "is" }, { "Len": 5, "NormalizedToken": "sharp", "Offset": 47, "RawToken": "sharp" }, { "Len": 3, "NormalizedToken": "and", "Offset": 53, "RawToken": "and" }, { "Len": 6, "NormalizedToken": "bright", "Offset": 57, "RawToken": "bright" }, { "Len": 3, "NormalizedToken": "but", "Offset": 64, "RawToken": "but" }, { "Len": 3, "NormalizedToken": "the", "Offset": 68, "RawToken": "the" }, </pre>	<pre> { "Len": 5, "NormalizedToken": "store", "Offset": 72, "RawToken": "store" }, { "Len": 4, "NormalizedToken": "does", "Offset": 78, "RawToken": "does" }, { "Len": 3, "NormalizedToken": "not", "Offset": 83, "RawToken": "not" }, { "Len": 4, "NormalizedToken": "have", "Offset": 87, "RawToken": "have" }, { "Len": 3, "NormalizedToken": "the", "Offset": 92, "RawToken": "the" }, { "Len": 4, "NormalizedToken": "apps", "Offset": 96, "RawToken": "apps" }, { "Len": 1, "NormalizedToken": "I", "Offset": 101, "RawToken": "I" }, { "Len": 4, "NormalizedToken": "need", "Offset": 103, "RawToken": "need" }, { "Len": 1, "NormalizedToken": ".", "Offset": 107, "RawToken": "." }] }]</pre>
--	--	--

- This phone has a great battery
- The display is sharp and bright
- The store does not have the apps I need

Let's run these three parts individually using the Text Analytics API and see what comes back and how this lets you make better sense of the review. I'll now use the key phrase and the sentiment operations of the Text Analytics API and show the output of this API transformed into a powerful targeted analysis of the sentiment reflected in the text input.

I also need to remove the punctuation marks and any conjunctions in order to get the best analysis. I changed my Curl script to

Figure 6 Using the Text Analytics API on the Output of the Linguistic Analysis API

```
{
  "documents": [
    {
      "keyPhrases": [
        "great battery"
      ],
      "id": "1"
    }
  ],
  "errors": []
}
{
  "documents": [
    {
      "score": 0.814330127882407,
      "id": "1"
    }
  ],
  "errors": []
}
{
  "documents": [
    {
      "keyPhrases": [
        "display"
      ],
      "id": "1"
    }
  ],
  "errors": []
}
{
  "documents": [
    {
      "score": 0.94089591967051,
      "id": "1"
    }
  ],
  "errors": []
}
{
  "documents": [
    {
      "keyPhrases": [
        "store",
        "apps"
      ],
      "id": "1"
    }
  ],
  "errors": []
}
{
  "documents": [
    {
      "score": 0.255424793209646,
      "id": "1"
    }
  ],
  "errors": []
}
}
```

pass the tokens one by one first to key phrase endpoint and then to the sentiment endpoint. **Figure 6** shows my result.

If I compile these results to a human-friendly format, here's what it looks like:

- This phone has a great battery
Key phrase: great battery
Sentiment: 81 percent
- The display is sharp and bright
Key phrase: display
Sentiment: 94 percent
- The store does not have the apps I need
Key phrase: store, apps
Sentiment: 25 percent

When discussing the Text Analytics API earlier in this article, I ran the complete review through the API and saw that the review had a 77 percent positive sentiment. However, running the review with the Linguistic API, breaking it down to its individual tokens and then processing them with the Text Analytics API gives a truer assessment of the individual aspects of the phone in question and the targeted sentiment analysis for each of them.

It also made clear that a better apps selection for this phone would have brought the overall positivity of this review much higher than the initial 77 percent.

Just as electricity became the
lifeblood of the second industrial
revolution, data is the lifeblood of
the AI revolution that's just ahead.

Wrapping Up

I recently came across the tagline, "Data is the new electricity." Just as electricity became the lifeblood of the second industrial revolution, data is the lifeblood of the AI revolution that's just ahead. As a developer, you gather data all the time, in various forms. One increasingly significant form of data is customer input—as reviews on shopping and traveling portals, in survey responses, feedback and suggestions both off- and online, and more. It's also becoming increasingly common for customers to openly provide such feedback, making it even more important to take this information seriously. Organizations will want to make sure to listen to their customers and take appropriate action to remediate any issues they encounter with products and services. The Text Analytics and the Linguistic Analysis Cognitive Services are just the tools you need to do justice to your customers. ■

ASHISH SAHU is a senior technical evangelist working with Developer Experience at Microsoft India, helping ISVs and startups overcome technical challenges, adopt the latest technologies, and evolve their solutions to the next level. He can be contacted at ashish.sahu@microsoft.com.

THANKS to the following Microsoft technical expert for reviewing this article:
Sandeep Alur

We Made WPF GREAT!



Xceed
Business Suite
for WPF

Match the vision you have for your WPF application's user experience, and surpass corporate expectations.



Azure Machine Learning Time Series Analysis for Anomaly Detection

Dawid Borycki

Anomaly Detection is one of the most important features of Internet of Things (IoT) solutions that collect and analyze temporal changes of data from various sensors. In many scenarios, sensor data doesn't change significantly over time. However, when it does, it usually means that your system has encountered an anomaly—and this anomaly can lead to a specific malfunction. In this article I'll show you how to use Azure Machine Learning Time Series Anomaly Detection to identify anomalous sensor readings. To this end I'll extend the RemoteCamera Universal Windows Platform (UWP) app I developed in my previous article (msdn.com/magazine/mt809116) by adding a list that displays anomalous values

This article discusses:

- Using Azure Machine Learning for time-series processing
- Creating and configuring machine learning models for anomaly detection
- Publishing machine learning algorithms as Web services
- Implementing a UWP app identifying anomalies with machine learning

Technologies discussed:

Universal Windows Platform, Azure Machine Learning, REST

Code download available at:

msdn.com/magazine/1117magcode

(see **Figure 1**). The RemoteCamera app acquires images from the webcam and calculates their brightness, which fluctuates around some specific value unless the camera image changes significantly. Because you can easily induce serious brightness changes (by covering the camera, for example), leading to irregularities, this app provides good input for time-series anomaly detection.

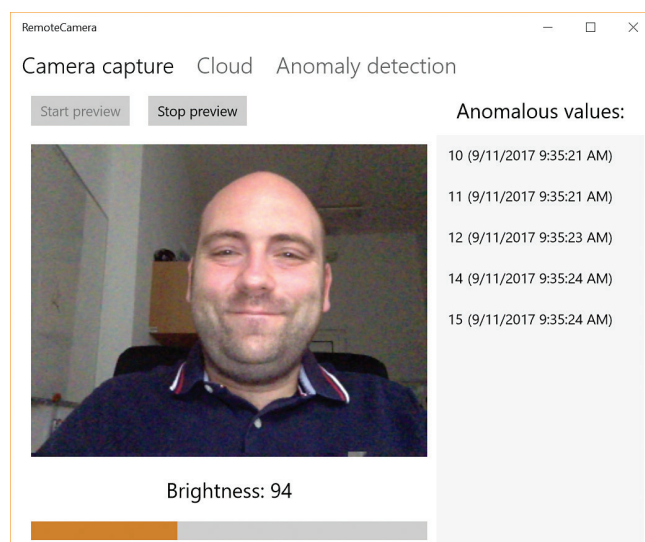


Figure 1 Detecting Anomalous Brightness Values with Azure Machine Learning

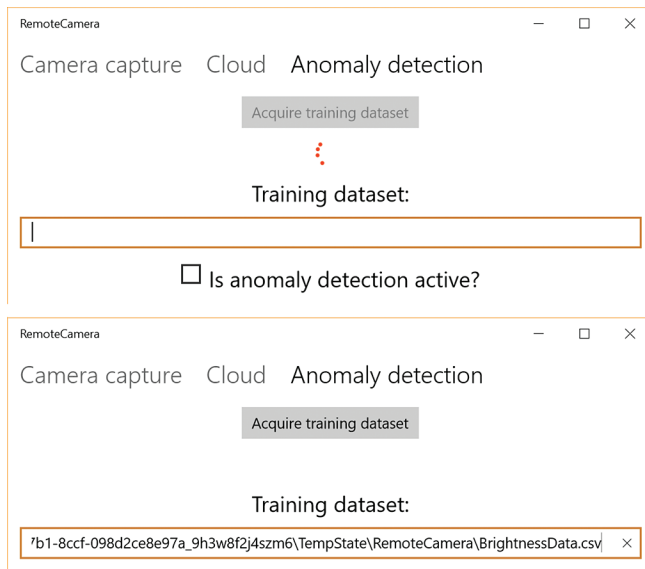


Figure 2 Anomaly Detection Tab of the RemoteCamera App

Anomaly Detection

As explained in a recent article by James McCaffrey (msdn.com/magazine/mt826350), one common way of detecting abnormalities is through time-series regression. By fitting a model to your data, you can predict trends and then see if all sequence values follow them by calculating the difference between actual and predicted values. A large divergence from expected values indicates outliers or anomalous values. Here, I'll first demonstrate how to detect such outliers by analyzing the so-called z-scores. The larger the z-score, the higher the probability that the actual value is an outlier. So, to find abnormalities, you specify the range of z-scores, which are treated as "normal." All z-scores outside that range indicate abnormalities. However, this approach uses a fixed threshold, thus it might lead to a large number of false positives. To solve such an issue, more-complex algorithms are employed. Specifically, the Azure Time Series Anomaly Detection module is based on exchangeability martingales (bit.ly/2wjBYUU), which analyze if a

Figure 3 Acquiring Training Dataset

```
private const int trainingDataSetLength = 100;
private List<BrightnessDataPoint> trainingDataSet =
    new List<BrightnessDataPoint>();

public event EventHandler<TrainingDataReadyEventArgs> TrainingDataReady;

public async Task AddTrainingValue(byte brightness)
{
    trainingDataSet.Add(new BrightnessDataPoint(brightness));

    // Check if all data points were acquired
    if (trainingDataSet.Count == trainingDataSetLength)
    {
        // If so, save them to csv file
        var brightnessFileStorage =
            await BrightnessFileStorage.CreateAsync();
        await brightnessFileStorage.WriteData(trainingDataSet);

        // ... and inform listeners that the training data set is ready
        TrainingDataReady?.Invoke(this,
            new TrainingDataReadyEventArgs(
                brightnessFileStorage.FilePath));
    }
}
```

sequence of values can be arbitrarily reordered without changing the probability of finding a given value in that sequence (or, in other words, that each value is equally likely to be found in a dataset). This exchangeability property of the dataset leads to small anomaly scores. When exchangeability is broken, large anomaly scores will be generated, indicating abnormal values.

In this article I'll demonstrate how to create such machine learning (ML) algorithms. I'll use the Microsoft Azure Machine Learning Studio (studio.azureml.net), which was also described by McCaffrey in the September 2014 issue (msdn.com/magazine/dn781358). Here, I'll go beyond that article and, as well as creating ML experiments, I'll show how to deploy the resulting solution as a Web service, then how to use such a service in the RemoteCamera app.

Training the Data Set

The first step is to extend the RemoteCamera app by adding another tab, which lets you acquire the training dataset and enable or disable anomaly detection using a checkbox (Figure 2).

The button, Acquire training dataset, becomes enabled after you start the camera preview (using controls from the first tab). When you tap this button, an app starts acquiring the training dataset. This works in the background and is indicated with a progress ring. The resulting training dataset comprises 100 data points, each of which is represented by an instance of the BrightnessDataPoint structure:

```
public struct BrightnessDataPoint
{
    public DateTime Time { get; private set; }
    public byte Brightness { get; private set; }

    public BrightnessDataPoint(byte brightness)
    {
        Time = DateTime.Now;
        Brightness = brightness;
    }
}
```

The BrightnessDataPoint struct stores a brightness value along with the time the brightness was determined. A collection of such values is then exported to the BrightnessData.csv file, which looks like the following:

```
Time,Brightness
9/8/2017 11:30:00,103
9/8/2017 11:30:01,103
9/8/2017 11:30:02,102
9/8/2017 11:30:03,42
9/8/2017 11:30:04,46
9/8/2017 11:30:05,149
9/8/2017 11:30:06,99
9/8/2017 11:30:07,101
9/8/2017 11:30:08,104
```

The particular location of the training dataset is then displayed in the text box. I use a comma-separated (CSV) file so it can be easily uploaded to the Machine Learning Studio.

To implement this functionality, I wrote two classes: BrightnessFileStorage and AnomalyDetector. The first class, BrightnessFileStorage, is defined in the BrightnessFileStorage.cs file in the AnomalyDetection subfolder of the companion code. BrightnessFileStorage saves a collection of BrightnessDataPoint objects to the CSV file using the DataWriter class (bit.ly/2wS31dq).

The second class, AnomalyDetector, handles the logic related to anomaly detection. In particular, it has a public method, AddTrainingValue, shown in Figure 3, which is invoked right after the image brightness is calculated (see the ImageProcessor_ProcessingDone

event handler in the MainPage.xaml.cs in the accompanying code). AddTrainingValue proceeds as follows: First, I create an instance of the BrightnessDataPoint, which is then added to the collection. When this collection has 100 items, I save it to the CSV file. I then fire the TrainingDataReady event, which is handled in MainPage to break training-dataset acquisition and display the file location in the UI:

```
private async void AnomalyDetector_TrainingDataReady(
    object sender, TrainingDataReadyEventArgs e)
{
    await ThreadHelper.InvokeOnMainThread(() =>
    {
        remoteCameraViewModel.IsTrainingActive = false;
        remoteCameraViewModel.TrainingDataSetFilePath = e.FilePath;
    });
}
```

The location of the training dataset is displayed in the textbox, so you can easily copy it and paste it in Windows Explorer to see the resulting data.

The z-Score Analysis

With the training dataset ready, I prepare the first experiment in Machine Learning Studio, following the instructions in McCaffrey's 2014 article. I first upload the BrightnessData.csv file, and then design the experiment using the visual designer, as shown in **Figure 4**. Briefly, all the components are in the menu, located on the left-hand side of the Machine Learning Studio. To place an element in your experiment, you simply drag it on the experiment

pane (the center part of the Machine Learning Studio). Each component has specific inputs and outputs. You connect compatible nodes to control the data flow between modules. Components can have additional configuration, which you set using the properties window (it appears on the right of the Machine Learning Studio).

The ML algorithm depicted in **Figure 4** works in two modes: experiment and Web service. They differ only in the input. In experiment mode, an input is composed of the uploaded training dataset (BrightnessData), which is replaced in the Web service mode by the Web service input. Independent of the mode, the input is converted to a dataset, then the values from the brightness column are normalized using the z-score transformation (bit.ly/2eWwHAA). The transformation converts brightness values to z-scores, which tell you how far the current value is from the mean. This distance is measured in standard deviations. The larger a distance, the higher the probability that the current value is an outlier. I apply the z-score normalization because, in general, the base or normal brightness level varies depending on what the camera sees. Thus, the z-score transformation ensures the correct brightness level, after normalization is close to 0. The raw brightness values vary from approximately 40 to 150. After normalization, all brightness values will fall between approximately -4.0 and +4.0, as shown in **Figure 5**. Consequently, to find anomalous values all I need to do is apply the threshold filter. Here, I use the Azure Machine Learning

Threshold Filter of type OutOfRange with lower and upper boundaries set to -2 and 1.5. I choose these values based on the z-scores plot in **Figure 5** and set them using the properties pad of the Threshold Filter in Machine Learning Studio.

After thresholding, the dataset contains one Boolean column, specifying whether a given time point is outside the specified range. To supplement this information with actual brightness values that are identified as outliers, I combine this column with the original dataset and then split the resulting dataset into two subsets: one containing anomalous values only and the other with normal values (see the bottom part of **Figure 4**). I change the column datatype before splitting because the Split Data module doesn't accept Boolean values. Then, the first subset is returned by the experiment. In the Web service view, this result is transferred to the client. Note that to see values from any dataset you use the Results dataset | Visualize option from the dataset context menu in Machine Learning Studio. This option works provided you've previously run the experiment. **Figure 6** depicts an example of such visualization of the last dataset from the experiment shown in **Figure 4**.

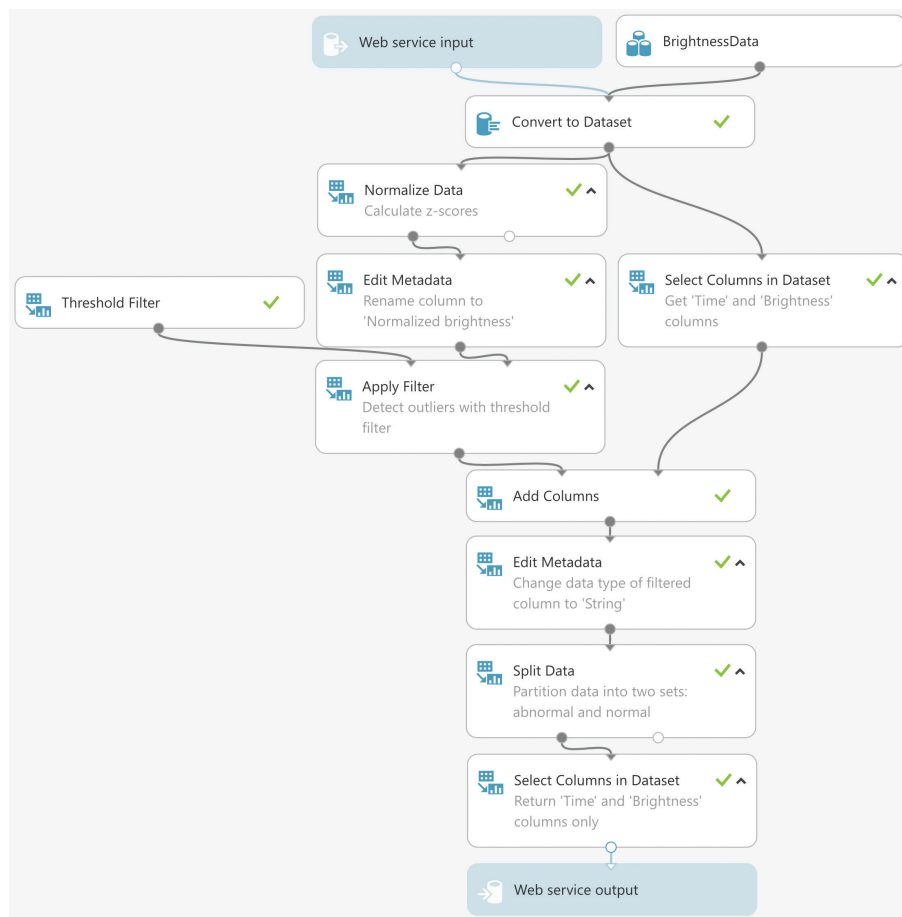


Figure 4 Anomaly Detection Using z-Score Analysis

Empower your development. Build better apps.

For more information, visit GrapeCity.com or call +1-800-240-7500

Achieve more with GrapeCity Developer Solutions.

Developers seek out GrapeCity and our great line of products because we get the fast-moving pace of the developer world. We live, breathe, and work within the developer universe. Their community is our community.

We're trusted partners, problem solvers, and team members.

Because GrapeCity is comprised of developers, we understand the unique challenges developers and their enterprises face. Our team and tools empower development by decreasing time, money, and security risks so the focus remains on developing innovative business solutions. GrapeCity's high-quality, fast, flexible UI controls and solutions have small footprints, key features, and universal APIs – all designed to reduce the learning curve and enhance the end product. And with direct access to global product experts, developers are never alone in the goal to develop meaningful, intuitive applications for enterprises and beyond.



ComponentOne
NET UI CONTROLS



ActiveReports
REPORTING SOLUTIONS



Spread
SPREADSHEET SOLUTIONS



Wijmo
JAVASCRIPT UI CONTROLS

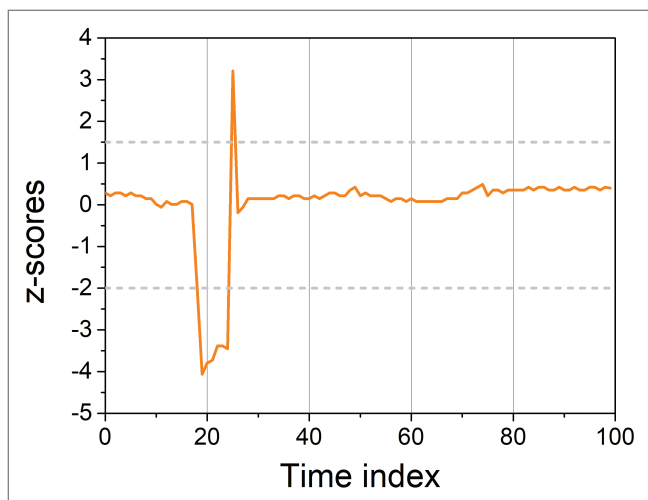


Figure 5 The Training Dataset After Normalization

Machine Learning Time-Series Analysis

Let's now see how to use the Azure Time Series Anomaly Detection (ATSAD) module to identify outliers. As shown in **Figure 7**, the flow of the experiment is quite similar to the previous one. The initial dataset is normalized with z-score transformation and transferred to the ATSAD module (you can find it under the Time Series node of Machine Learning Studio). This requires you to provide several inputs, which you configure with the properties window (bit.ly/2xUGtg2). First, you specify the data and time columns, then you configure martingale type. Here, I use the Power martingale. This activates another textbox, Epsilon, in which you can type any value from 0 to 1 to specify the sensitivity of the detector. Then, you choose a strangeness function, using one of three options:

- **RangePercentile**: Use this option to identify values that are clearly outside the range, like spikes or dips. I use this option in my experiment so it will work analogously to the previous experiment, but with a more comprehensive analysis.
- **SlowPos- and SlowNegTrend**: Use these options to identify positive and negative trend changes in your dataset. This is useful when your solution looks for increases or decreases in observed values.

Next, you specify the length of the martingale and strangeness values history. You can choose any integer between 10 and 1000.

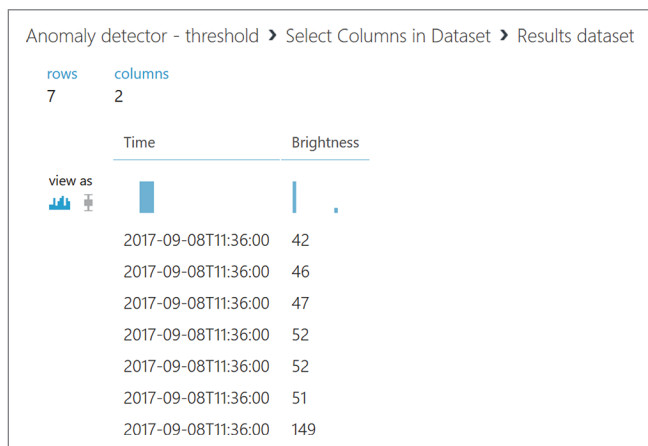


Figure 6 Anomalous Values Detected with z-Score Analysis

After a bit of trial-and-error experimentation, I settled on the following parameters for my detector:

- Epsilon = 0.4
- Length of martingale and strangeness values history = 50

The last parameter of the detector is an alert threshold, which specifies the minimum value of the anomaly score that marks the given value as an outlier. By default, the alert threshold is set to 3.5. For my experiment, I changed this to 2.

If you visualize the output of the ATSAD module, you'll see that it supplements the input dataset with two columns: the anomaly score, which measures abnormalities, and the alert indicator, which contains a binary value (0 or 1) indicating if a value is anomalous. I use the latter to split the dataset into two subsets: normal and abnormal. Only the abnormal subset is returned by the experiment. The other elements of the experiment are the same as before so I won't discuss them again. I'll only note that a very important aspect of the experiment required for a Web service is the name of the input and output. I set these values to Data (Web service input) and AnomalyDetectionResult (Web service output).

Web Service Client

With the experiments set up I can now publish them as Web services so they can be accessed by the RemoteCamera app to identify any image brightness abnormalities. To set up a Web

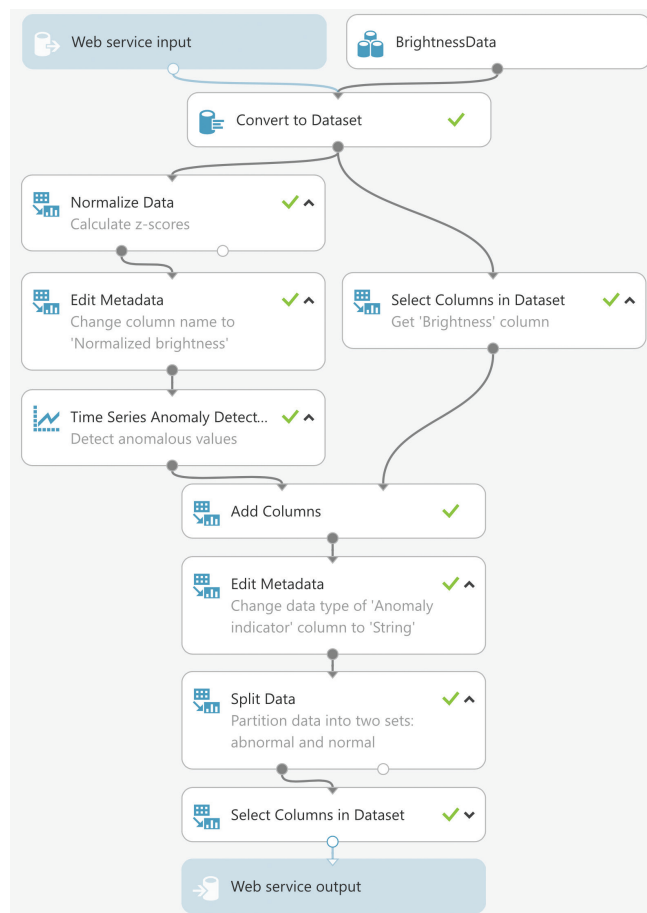


Figure 7 An Experiment Utilizing the Azure Time Series Anomaly Detection Module

service you need to run the experiment and then press the Deploy Web Service icon on the bottom pane of Machine Learning Studio (see the highlighted item in **Figure 8**). If you don't add Web service input and output modules to the experiment, this pane will show Set Up Web Service. If you click it, Web service input and output modules will be added to the experiment, and the button label will change to Deploy Web Service.

When the Web service deployment completes, you'll be redirected to the Web service dashboard, shown in **Figure 9**. The dashboard presents a summary of the published Web service, along with the API key and instructions on how to send requests and handle responses (API Help Page). Specifically, after clicking the request/response hyperlink, a Web service URL and a detailed request and response structure in JSON format will be presented.

To proceed further, I store the API key and create JSON-to-C# mappings using the JSONUtils service (jsonutils.com). I then save the resulting classes in corresponding files, AnomalyDetectionRequest.cs and AnomalyDetectionResponse.cs, in the AnomalyDetection sub-folder. Their structure is similar—both files contain classes, which as usual are composed mostly of the auto-implemented properties. AnomalyDetectionRequest and AnomalyDetectionResponse represent corresponding JSON objects being transmitted between a client and a Web service. As an example, the definition of the AnomalyDetectionRequest class and dependent objects is given in **Figure 10**. Note that to convert a collection of brightness data points to an input accepted by the Machine Learning Studio Web service (two dimensional array of strings), I use a helper class, ConversionHelper. The latter, which full definition is in the companion code, has two public methods. They either convert the collection of brightness data points to string[,] (BrightnessDataToStringTable) or vice versa (AnomalyDetectionResponseToBrightnessData).

Once the JSON-to-C# object mapping is established, I can write the actual Web service client. To that end I first install the Microsoft.AspNet.WebApi.Client NuGet package and then use it to define the AnomalyDetectionClient class (see the corresponding file in the companion code). This class has three private fields: baseAddress, apiKey and httpClient. The first field stores the URL of the Machine Learning Studio Web service, while the second contains the API key. These two values are used to instantiate the HttpClient class (from the installed NuGet package):

```
public AnomalyDetectionClient()
{
    httpClient = new HttpClient()
    {
        BaseAddress = new Uri(baseAddress),
    };

    httpClient.DefaultRequestHeaders.Authorization =
        new AuthenticationHeaderValue("Bearer", apiKey);
}
```

After creating the client, I can start sending requests to the Machine Learning Studio Web service with the AnomalyDetectionClient.DetectAnomalyAsync method from **Figure 11**. This method accepts a collection of brightness data points, representing test data. This test data replaces the CSV file I used previously for experimenting and is used to instantiate AnomalyDetectionRequest. An instance of this class is later posted to the Web service

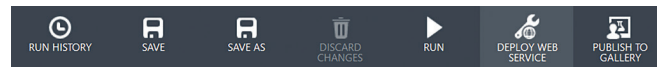


Figure 8 The Azure Machine Learning Studio Action Pane

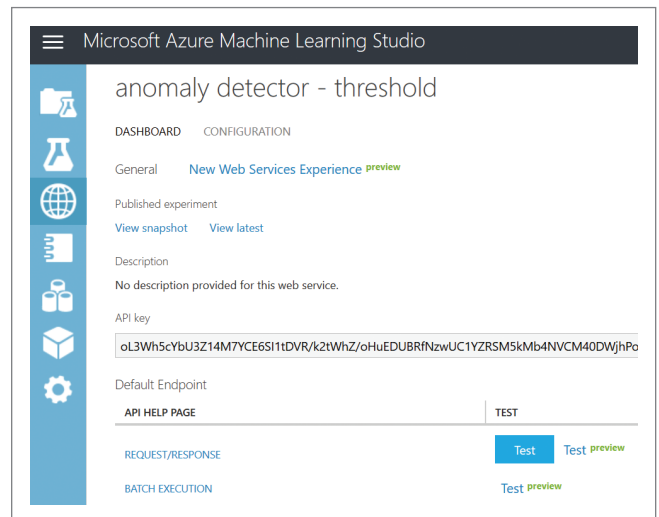


Figure 9 The Web Service Dashboard

for analysis with the PostAsJsonAsync extension method. The resulting JSON response is converted to the AnomalyDetectionResponse class instance, which is finally returned by the DetectAnomalyAsync function. I also look for any errors and eventually throw an exception if appropriate.

Figure 10 A Definition of the AnomalyDetectionRequest Class and Dependent Objects

```
public class AnomalyDetectionRequest
{
    public Inputs Inputs { get; set; }
    public GlobalParameters GlobalParameters { get; set; }

    public AnomalyDetectionRequest(
        IList<BrightnessDataPoint> brightnessData)
    {
        Inputs = new Inputs()
        {
            Data = new Data()
            {
                ColumnNames = new string[]
                {
                    "Time",
                    "Brightness"
                },
                Values = ConversionHelper.
                    BrightnessDataToStringTable(brightnessData)
            }
        };
    }
}

public class Inputs
{
    public Data Data { get; set; }
}

public class Data
{
    public string[] ColumnNames { get; set; }
    public string[,] Values { get; set; }
}

public class GlobalParameters { }
```

WE ARE CHANGING THE WAY YOU LOGIC REPORTING



WINDOWS FORMS



WPF



ASP.NET MVC



ASP.NET AJAX



CLOUD WEB API

www.textcontrol.com

ING OOK AT

TEXT CONTROL



The `AnomalyDetectionClient` is utilized in the `AddTestValue` method of the `AnomalyDetector` class (**Figure 12**). Like `AddTrainingValue`, `AddTestValue` is also invoked in the `ImageProcessor_ProcessingDone` event handler (see `MainPage.xaml.cs` in the companion code). However, `AddTestValue` proceeds in a slightly different manner than the `AddTrainingValue` method. In `AddTestValue` I add brightness data points to an instance of the `BrightnessDataset` class, which internally uses the generic `List` class to implement a rolling window. This window, like the one in James McCaffrey's October article, is used to store test values. By default, the size of the rolling window is set to 30 elements, but you can control this value using a constructor of the `BrightnessDataset`. As shown in **Figure 12**, I don't send data for analysis until the window is full, then I check whether the collection of anomalous values returned by the Web service contains any elements. If so,

Figure 11 Sending Requests to the Azure Machine Learning Studio Web Service

```
public async Task<IList<BrightnessDataPoint>>
DetectAnomalyAsync(IList<BrightnessDataPoint> brightnessData)
{
    var request = new AnomalyDetectionRequest(brightnessData);

    var response = await httpClient.PostAsJsonAsync(string.Empty, request);

    IList<BrightnessDataPoint> result;

    if (response.IsSuccessStatusCode)
    {
        var anomalyDetectionResponse = await
            response.Content.ReadAsAsync<AnomalyDetectionResponse>();

        result = ConversionHelper.
            AnomalyDetectionResponseToBrightnessData(anomalyDetectionResponse);
    }
    else
    {
        throw new Exception(response.ReasonPhrase);
    }

    return result;
}
```

Figure 12 Detecting Anomalies

```
public event EventHandler<AnomalyDetectedEventArgs> AnomalyDetected;
private BrightnessDataset dataSet = new BrightnessDataset();

public async Task AddTestValue(byte brightness)
{
    dataSet.Add(new BrightnessDataPoint(brightness));

    if (dataSet.IsFull)
    {
        try
        {
            var anomalousValues = await anomalyDetectionClient.
                DetectAnomalyAsync(dataSet.Data);

            if (anomalousValues.Count > 0)
            {
                AnomalyDetected?.Invoke(this,
                    new AnomalyDetectedEventArgs(anomalousValues));
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine(ex);
        }
    }
}
```

I invoke the `AnomalyDetected` event, which is also used to pass abnormalities to listeners.

To display anomalous values in the UI, I handle the `AnomalyDetected` event in the `MainPage` class as follows:

```
private async void AnomalyDetector_AnomalyDetected(
    object sender, AnomalyDetectedEventArgs e)
{
    await ThreadHelper.InvokeOnMainThread(() =>
    {
        foreach (var anomalousValue in e.AnomalousValues)
        {
            if (!remoteCameraViewModel.AnomalousValues.Contains(anomalousValue))
            {
                remoteCameraViewModel.AnomalousValues.Add(anomalousValue);
            }
        }
    });
}
```

Specifically, I iterate over the collection of obtained values to check whether they were already added to the local datastore (the `AnomalousValues` property of the view model). If not, I add them to the observable collection. As a result, only new abnormal values will appear in the list shown previously in **Figure 1**. I do this additional check because of the rolling window, in which only one element is changed between successive calls to the Web service.

To test my solution, you'll need to run the `RemoteCamera` app, start the camera preview and enable anomaly detection using the checkbox on the `Anomaly Detection` tab. Once this is done you can generate abnormal values by covering your camera. These values should be quickly recognized by the remote ML detector as anomalous and displayed in the listbox (as in **Figure 1**).

By default, the size of the rolling window is set to 30 elements, but you can control this value using a constructor of the `BrightnessDataset`.

Wrapping up

I demonstrated here how to design two different anomaly detection experiments in the Azure Machine Learning Studio. Both experiments were also deployed as Web services and combined with the client app, `RemoteCamera`, which sends locally acquired time-series data for machine learning analysis to identify abnormalities. Here, I used a Web service in the UWP app. However, you can use the same code to access a Web service from an ASP.NET Web app, where you handle the ML logic at the back end rather than on the endpoint—which, in the case of IoT can be just a simple sensor. ■

DAWID BORYCKI is a software engineer and biomedical researcher, author and conference speaker. He enjoys learning new technologies for software experimenting and prototyping.

THANKS to the following Microsoft technical expert for reviewing this article:
Dr. James McCaffrey

Manipulating Documents?

APIs to view, convert, annotate, compare, sign, assemble and search documents in your applications.

Try GroupDocs APIs for FREE



GroupDocs.Viewer

View over 50 documents and image formats in any application using document viewer APIs.



GroupDocs.Annotation

Add annotations to specific words, phrases and any region of the document.



GroupDocs.Conversion

Fast batch document conversion APIs for any .NET, Java or Cloud app.



GroupDocs.Comparison

Compare two documents and get a difference summary report.



GroupDocs.Signature

Digitally sign Microsoft Word, Excel, PowerPoint and PDF documents.



GroupDocs.Assembly

Document automation APIs to create reports from templates and various data sources.

▶ GroupDocs.Metadata

▶ GroupDocs.Search

▶ GroupDocs.Text

▶ GroupDocs.Editor



Americas: +1 903 306 1676
EMEA: +44 141 628 8900
Oceania: +61 2 8006 6987
sales@poseptyltd.com



Download a Free Trial at
<https://downloads.groupdocs.com/>

Securing Data and Apps from Unauthorized Disclosure and Use

Joe Sewell

“Data breach.” These two words are the scariest words in software development. Software is all about the data—processing it, managing it and securing it. If an attacker can breach that data, your business could forfeit confidential information critical to your success, be subject to liability nightmares, and lose valuable brand respect and customer loyalty.

To manage this risk and comply with regulations like HIPAA and GDPR, wise developers and operations teams will implement

security controls on databases and Web services. These controls include end-to-end encryption, rigorous identity management, and network behavior anomaly detection. In the event of a security incident, many such controls will react *actively* by recording incident details, sending alerts and blocking suspicious requests.

While these and other best practices can secure your server components, they don’t do as much for client software. And for your data to be useful, it must be exposed in some form to privileged users and software. How can you also be sure that your client software doesn’t cause a data breach?

For instance, many companies create software specifically for use by their employees, often designed to access sensitive corporate data. And while the Microsoft .NET Framework makes it easy to develop line-of-business (LOB) apps in languages like C# or Visual Basic .NET, those compiled apps still contain high-level metadata and intermediate code. This makes it easy for a bad actor to manipulate the app with unauthorized use of a debugger, or to reverse engineer the app and create a compromised version. Both scenarios could lead to a data breach, even if the server components are completely secure.

While there are some measures you can take to guard against these attacks—Authenticode signing and code obfuscation, to name two—most of them are *passive* in that they merely deter attacks, rather than detect, report and respond to them. But, recently, new features in Visual Studio allow you to inject threat detection,

This article relies on a preview version of Dotfuscator Community Edition version 5.32. All information is subject to change.

This article discusses:

- Why clients that deal with sensitive data must be secured
- How an attacker can use debugging tools to compromise such clients
- How to use Runtime Checks to protect clients
- How Runtime Checks fit into a broader, layered approach to security

Technologies discussed:

Runtime Checks, Dotfuscator Community Edition, Visual Studio 2017

Code download available at:

bit.ly/2yiT2eY

reporting and response capabilities into your .NET apps, with little-to-no additional coding required. These Runtime Checks are an active protection measure, able to change your app's behavior in response to a security threat, thus protecting your sensitive data.

In this article, I present a methodology for using Runtime Checks effectively, using a typical LOB app as an example. I'll explore how an attacker can use a debugger to cause a data breach, detail how Runtime Checks can protect against this breach and discuss the role of these controls in a layered-protection approach.

Sample Solution

To help explain how a data breach can happen in a client, I've prepared a sample LOB solution that can be downloaded from bit.ly/2yIT2eY. Instructions for building, running and using the various parts of the solution are available in the sample code's README.

The solution has four components:

AdventureWorks2014 Database: This is a Microsoft SQL Server database containing the Adventure Works 2014 OLTP database sample. You can download that sample from bit.ly/2wM0yy0.

Adventure Works Sales Service: This is an ASP.NET Web service that exposes customer data from the database, including sensitive data like credit cards. In order to make this component easier to set up, the sample code omits most security controls, but for the purposes of this article I'll assume the service implements the following:

- Two factors of authentication—a user password and an SMS sent to the user's phone on login
- Time-limited sessions
- SSL encryption for all requests and responses

Adventure Works Sales Client: This is a Windows Presentation Foundation (WPF) desktop client that connects to the Sales Service to manipulate the customer data. This is the component with which the article will be most concerned.

When a Sales employee runs the Client, they log in through the LoginDialog, which starts the authenticated session and opens the CustomerWindow. From this window, the employee can view and edit customer names, or open EmailWindow, PhoneWindow, or CreditCardWindow to edit a customer's sensitive data. Some common functions are also provided in a static class named Utilities.

Application Insights: While not required to run the sample, both the Sales Service and Client can send usage and error telemetry to Application Insights. With the Runtime Checks discussed in this article, the Client's telemetry also includes security incident reporting.

For this article, I'll be focusing on securing the Sales Client. I'll assume the database and Sales Service are already secured. Of course, that's not a safe assumption to make in a real scenario, but it helps demonstrate a point: even if you "do everything right" with server security, data breaches are still possible through client software.

I will also treat customer names as non-sensitive data and instead focus on securing

e-mail addresses, phone numbers and credit cards. In a real scenario, customer names would also be considered sensitive, and the non-sensitive data could include things such as retail store addresses.

Data Breach with a Debugger

Debuggers are wonderful development tools. They allow you to discover critical logic errors, step through tricky control-flow scenarios and diagnose crash dumps. However, like any tool, debuggers can also be used for evil.

Let's say that the Adventure Works intranet has a bad actor—perhaps a vindictive employee, an outside contractor, or even an external attacker who's gained unauthorized access to the intranet. This attacker doesn't have access to the database or Sales Service, but they can access a Sales employee's laptop. Certainly, this is a security problem, but because the Sales Service implements two-factor authentication and the attacker doesn't have access to the employee's phone, the customer data should be safe, right?

Actually, no. The attacker can wait for the Sales employee to log in through the Sales Client and then, either manually or through a script, attach a debugger to the Client process. Because the Client is a .NET app, the debugger will reveal a lot of high-level information, including the session token, even if no debugging symbols (PDB files, for example) are present.

Figure 1 demonstrates this scenario. Using the WinDbg debugger (bit.ly/2sh4clf) with the Psscor4 extension (bit.ly/2hbG2kk), I dumped various .NET objects in the memory of a running Sales Client process. I eventually found the AuthToken object and dumped the value of its HashField property.

With this session token, the attacker can make authenticated requests to the Sales Service in the employee's name. The attacker need not continue debugging or manipulating the Client; once he has the session token, he can go directly to the Web service and use the token to cause a data breach.

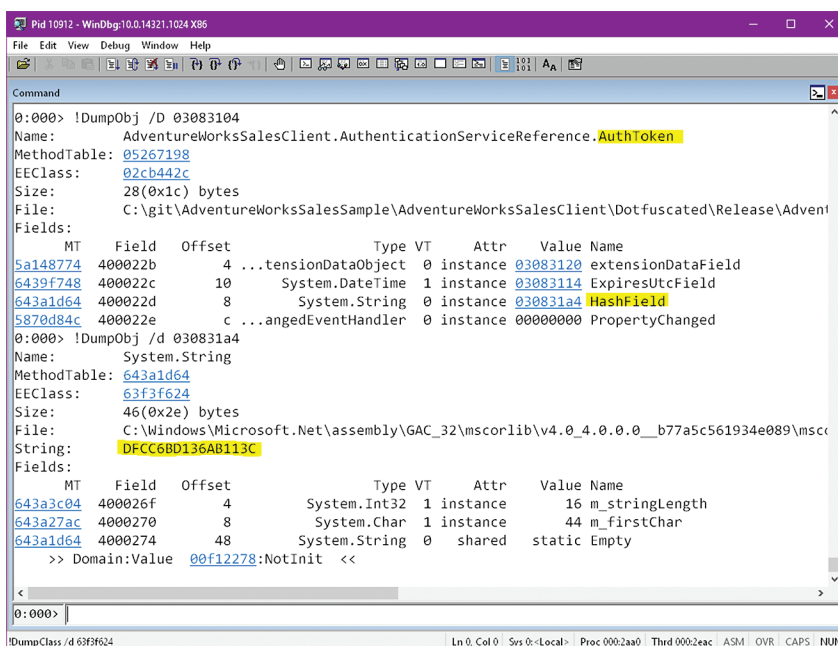


Figure 1 WinDbg Revealing a Session Token in the Sales Client

There are other scenarios where bad actors could use the Client in an unauthorized way:

Manipulating Sensitive Data Directly: While the preceding scenario was a session hijacking attack, because the Sales Client accesses sensitive data (such as credit cards) as part of its normal operation, that data can also be seen with a debugger. Attackers could even cause the app to behave unusually or modify the data in the database.

Reverse Engineering: Attackers could also run the Client themselves and attach a debugger to discover how the Client works. Combined with the ease of decompiling .NET apps, the attackers might be able to discover exploits or other important details about the Client or Service that would help them plan an attack.

Tampering: If attackers can reverse engineer the app and access the employee's file system, they can replace the legitimate Client with a modified one, secretly extracting or manipulating data when the employee logs in.

Other apps might be vulnerable to debuggers in different ways. For instance, an app that reports an employee's location for purposes of tracking fieldwork could be manipulated to provide inaccurate data. Or, a game might reveal key strategic information in a debugger.

About Runtime Checks

Runtime Checks is a new feature in PreEmptive Protection - Dotfuscator Community Edition (CE), a protection tool included with Visual Studio since 2003 (bit.ly/2wB0b9g). You may know that Dotfuscator CE can obfuscate the intermediate code of .NET assemblies, but obfuscation isn't the subject of this article. Instead, I'll be showing how I used Runtime Checks—hereafter, just Checks—to allow the Sales Client to protect itself while it runs.

Checks are prebuilt validations Dotfuscator can inject into your .NET apps. Your apps will then be able to detect unauthorized use, like debugging or tampering. Despite the name, Checks do more than just detect these states; they can also react in pre-specified ways, such as by exiting the app. Checks can also call into application code, allowing for custom behavior based on the Check's result. These reporting and response features are configurable per-Check, so all your apps can detect unauthorized uses in the same way, but each app can respond to that detection differently.

The code sample includes a Dotfuscator.xml config file that can instruct Dotfuscator to protect the Sales Client from unauthorized debugging and tampering. In the rest of this article, I'll explain how I created this configuration, what choices I made, and how you can similarly configure Dotfuscator to protect your own apps.

Setting Up the Tooling

The easiest way to get started with Dotfuscator CE is to use the Visual Studio Quick Launch (Ctrl+Q) to search for “dotfuscator.” If Dotfuscator isn't installed, an option to install PreEmptive Protection - Dotfuscator will appear; select that option and confirm the appropriate dialogs.

Once Dotfuscator is installed, repeating this search will provide an option to launch Tools | PreEmptive Protection - Dotfuscator; select that option to begin using Dotfuscator. After some typical first-time use dialogs, the Dotfuscator CE UI opens.

Important Note: The protection described here and included in the sample requires at least version 5.32 of Dotfuscator CE. You can see what version you have installed by choosing Help | About. If you're using an earlier version, please download the latest version of Community Edition from bit.ly/2fuUeow.

Checks are prebuilt validations
Dotfuscator can inject into your
.NET apps. Your apps will then be
able to detect unauthorized use.

Dotfuscator operates on specialized config files that specify what assemblies it should protect and how to apply protection. Dotfuscator starts with a new config file loaded; I adjusted it for the Sales Client using the following steps:

1. First, I saved the new config file as AdventureWorksSalesClient\Dotfuscator.xml.
2. Next, I told Dotfuscator where to find the Client's assemblies. I switched to the Dotfuscator Inputs screen and clicked the green plus-sign icon. From the Select Input browse dialog, I navigated to the AdventureWorksSalesClient\bin\Release directory and then clicked Open without selecting a file.
3. Dotfuscator added the whole directory as an input named Release. I expanded the tree node to verify that the AdventureWorksSalesClient.exe assembly was present.
4. Then, I made the config file portable, instead of specific to absolute paths in my environment. I selected the Release node, clicked the pencil icon and replaced the absolute path with \${configdir}\bin\Release. \${configdir} is a Dotfuscator macro that represents the directory holding the config file.
5. Finally, as this article isn't concerned with the Dotfuscator code obfuscation features, I disabled them by right-clicking on the Renaming item in the Dotfuscator navigation list and unchecking Enable.

Configuring Check Injection

Dotfuscator allows you to configure Checks from the Injection screen, on the Checks tab. What choices you make for the configuration, however, vary depending on the kind of app you're protecting. Rather than list all the features and settings, I'll walk-through the choices and configuration I made for the Sales Client sample.

For the sample, I configured three Checks:

- Two Debugging Checks, which detect unauthorized use of a debugger:
 - A “Login” Debugging Check, to detect session hijacking scenarios as described earlier
 - A “Query” Debugging Check, to detect a debugger being used to read/write sensitive data in the Client
- One Tamper Check, which detects use of a modified application binary

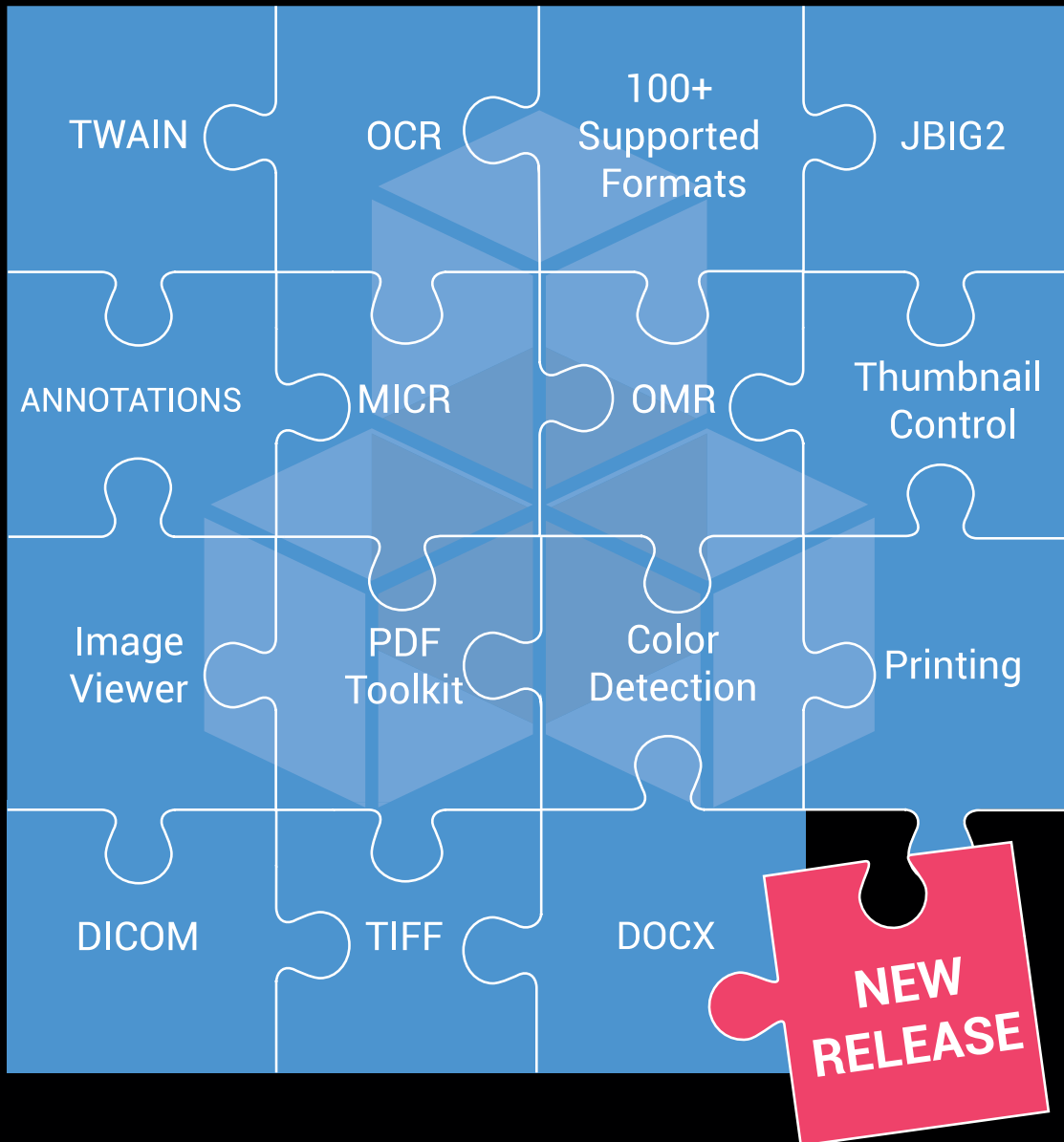
GdPicture.NET



14

100% ROYALTY FREE

Imaging SDK For WinForms, WPF And Web Development



Leverage your apps. with **GdPicture.NET** Imaging Toolkit

DOWNLOAD
YOUR FREE TRIAL

www.gdpicture.com

GdPicture.NET is an



product

Figure 2 shows a bird's-eye view of the Checks and the app code with which they interact. In the next three sections, I'll explain the purpose and configuration of each of these Checks.

Configuring the Login Debugging Check

This first Debugging Check addresses the session hijacking scenario. It detects if a debugger is present during the authentication process and notifies the app if so. The app will report the incident to Application Insights and then later error in unintuitive ways.

I added this Check by clicking on the Add Debugging Check button, which brought up a new configuration window. I configured this Check as seen in Figure 3.

Location: I first picked where in the app code the Check should run. As this Check should detect debugging when the user logs in, I checked the `LoginDialog.ConfirmLogin` method from the Locations tree.

Note that the Check only runs when its location is called. If an attacker attaches a debugger later, this Check won't detect it; but I'll address this issue later with the Query Debugging Check.

Application Notification: After a Check runs, it can notify the app code so the app can report and react in a customized way. The code element that receives this notification is known as the Application Notification Sink, and is configured using the following Check Properties:

- **ApplicationNotificationSinkElement:** The kind of code element (field, method and so on)
- **ApplicationNotificationSinkName:** The name of the code element
- **ApplicationNotificationSinkOwner:** The type that defines the code element

For all three Checks, I used this feature to report incidents to Application Insights. Additionally, for this Check, I decided that I wanted a custom response rather than a default response injected by Dotfuscator (which the other Checks will use). My response allows the login to succeed, but then crashes the app a few moments later. By separating the detection and response, I make it harder for an attacker to discover and work around the control.

To accomplish this response, I added a Boolean field, `isDebugged`, to the `LoginDialog` class and configured it as the Check's Sink. When the Check runs (that is, when the app calls `LoginDialog.ConfirmLogin`), the result of the debugger detection is stored in this field: true for a debugger detected, and false otherwise.

Note that the Sink must be accessible and writable from the Check's location. As both the location and Sink are instance members of the `LoginDialog` class, this rule is satisfied.

Next, I modified `LoginDialog.RunUserSession` to pass this field to the `CustomerWindow` constructor:

```
// In LoginDialog class
private void RunUserSession(AuthToken authToken)
{
    // ...
    var customerWindow = new Windows.CustomerWindow(clients, isDebugged);
    // ...
}
```

Then, I made the `CustomerWindow` constructor set its own field, `CustomerWindow.isDebugged`, and then report the incident to Application Insights:

```
// In CustomerWindow class
public CustomerWindow(Clients clients, bool isDebugged)
{
    // ...
    this.isDebugged = isDebugged;
    if (isDebugged)
    {
        // ClientAppInsights is a static class holding the Application
        // Insights telemetry client
        ClientAppInsights.TelemetryClient.TrackEvent(
            "Debugger Detected at Login");
    }
    // ...
}
```

Finally, I added code that reads this field to various event handlers. For instance:

```
// In CustomerWindow class
private void FilterButton_OnClick(object sender, RoutedEventArgs e)
{
    // ...
    if (isDebugged) { throw new InvalidCastException(); }
    // ...
}
```

I'll address the obviousness of the field name `isDebugged` later in this article.

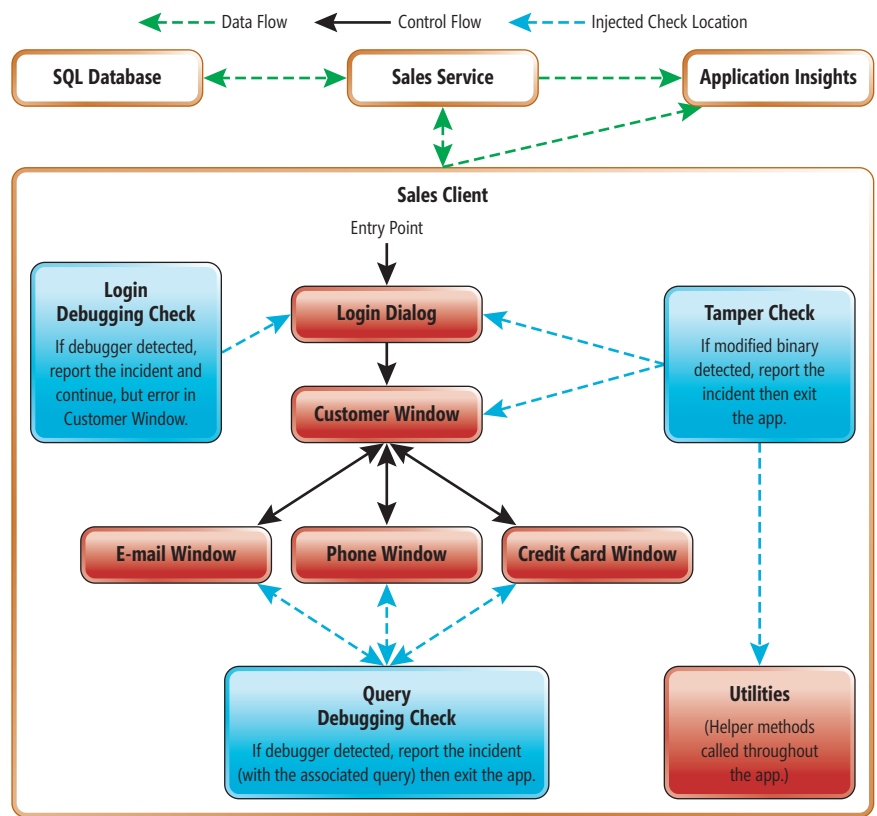
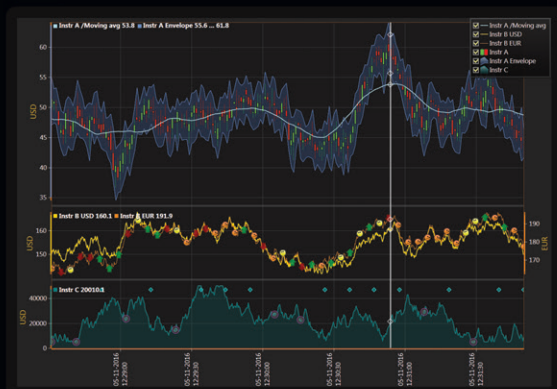
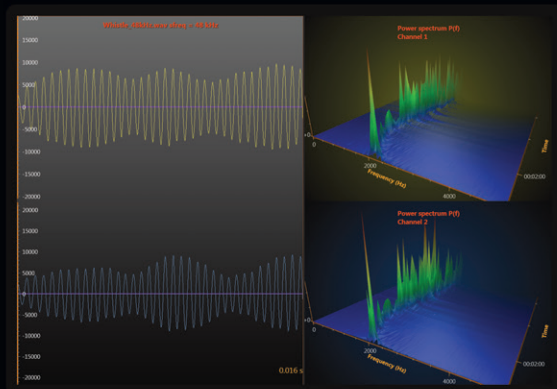
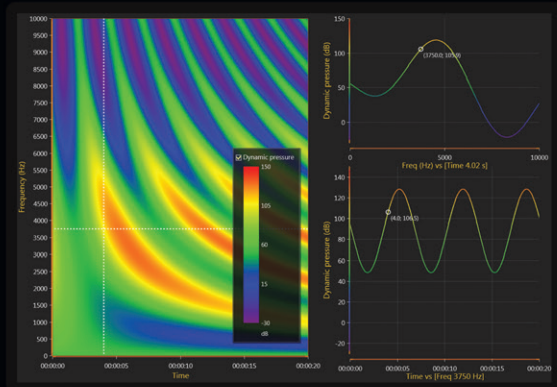


Figure 2 The Sales Client with Injected Runtime Checks



10% off*



LightningChart[®]

The fastest and most advanced charting components

Create **eye-catching** and **powerful** charting applications for engineering, science and trading

- DirectX GPU-accelerated
- Optimized for real-time monitoring
- Supports gigantic datasets
- Full mouse-interaction
- Outstanding technical support
- Hundreds of code examples
- Now with Volume Rendering extension
- Flexible licensing options

*10% off for Gold and Platinum packages until 12/31/2017!

Get free trial at
LightningChart.com/try



Configuring the Query Debugging Check

Because a debugger can be attached to the Client at any time during its execution, the Login Debugging Check alone is insufficient. The Query Debugging Check fills this gap by checking for a debugger when the app is about to query sensitive data, such as credit card numbers. The sensitivity of this data also means I can't afford to separate the detection, reporting and response as with the Login Debugging Check, because that would let an attacker see the data. Instead, the Query Debugging Check will report the incident and then immediately exit the app when a debugger is detected.

I added the second Debugging Check the same way I added the first one, but this time, I configured the Check as seen in Figure 4.

Locations: There are three kinds of sensitive data in my scenario: e-mail addresses, phone numbers and credit cards. Luckily, you can select multiple locations for a single Check. In this case, those locations are EmailWindow.UpdateData, PhoneWindow.UpdatePhones and CreditCardWindow.UpdateData. The Check runs whenever any of these are called, which means I have to configure only one set of Check Properties for all three kinds of sensitive data.

Application Notification: Having multiple locations modifies how the Application Notification Sink works. In the Login Debugging Check, I could specify the LoginDialog.isDebugged field as the Sink, because that field was accessible from the Check's only location, LoginDialog.ConfirmLogin. This time, each location must be able to access a Sink.

Notably, if the ApplicationNotificationSinkOwner property is blank, the Sink defaults to using the type that defines the Check's location. Because this Check has multiple locations, the Sink will thus vary depending on the location that triggered the Check. In this case, I left this property blank and set the other ApplicationNotificationSink properties to a method named ReportDebugging.

Consider the EmailWindow.ReportDebugging method:

```
// In EmailWindow class
private void ReportDebugging(bool isDebugging)
{
    if (isDebugging)
    {
        ClientAppInsights.TelemetryClient.TrackEvent(
            "Debugger Detected when Querying Sensitive Data",
            new Dictionary<string, string> { { "Query", "Email Addresses" } });
        ClientAppInsights.Shutdown();
    }
}
```

When the app calls the EmailWindow.UpdateData method, the Check runs and then calls this ReportDebugging method with

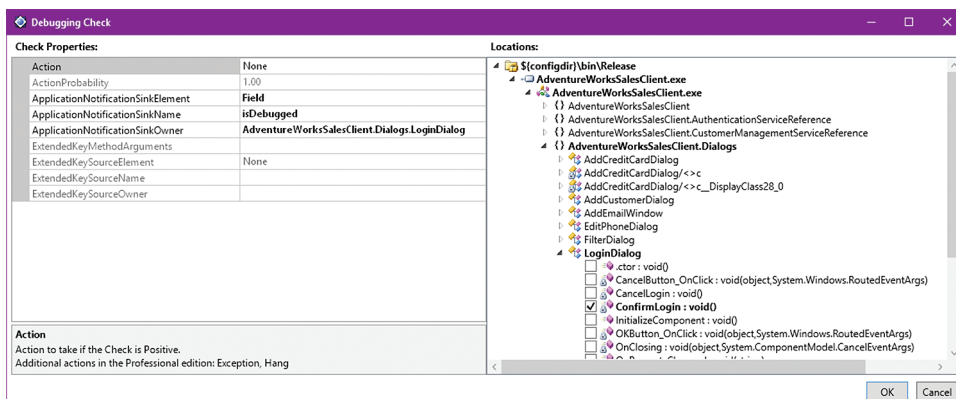


Figure 3 Configuration for the Login Debugging Check

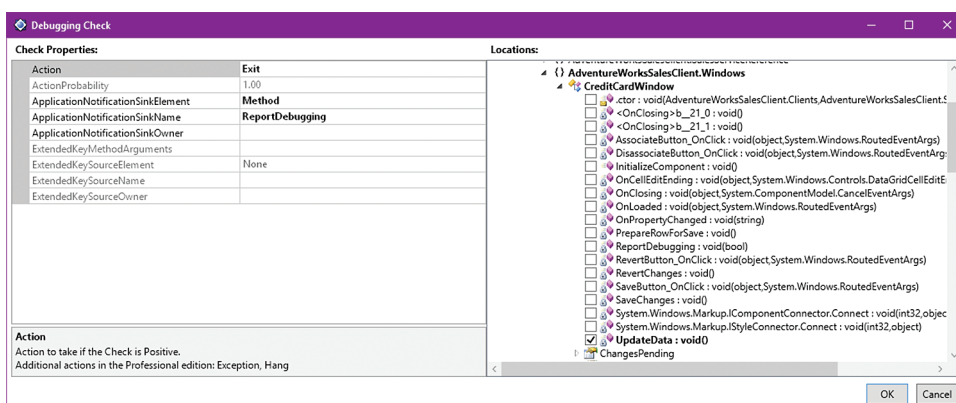


Figure 4 Configuration for the Query Debugging Check Showing the CreditCardWindow.UpdateData Location—Other Locations Aren't Shown

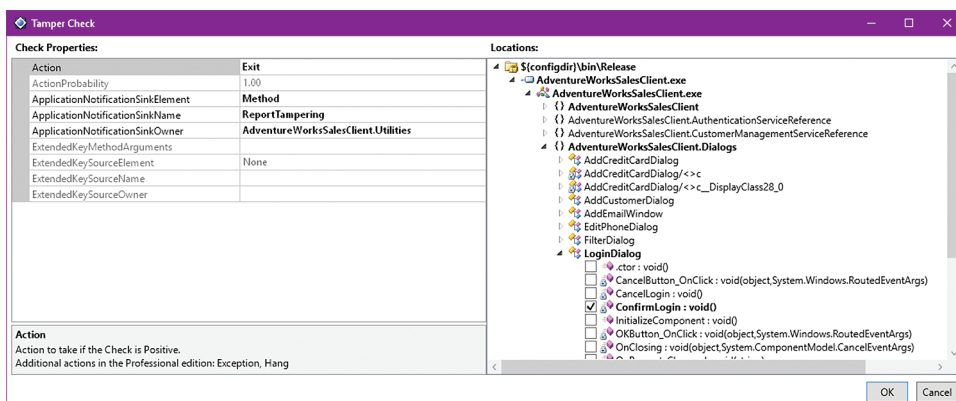


Figure 5 Configuration for the Tamper Check Showing the LoginDialog.ConfirmLogin Location—Other Locations Aren't Shown



Rider

New .NET IDE

**Cross-platform.
Killer code analysis.
Great for refactoring.**

From the makers of ReSharper,
IntelliJ IDEA, and WebStorm.

Learn more
and download
jetbrains.com/rider



JET
BRAINS

the argument true if debugging was detected and false otherwise.

The same thing happens when the app code calls `PhoneWindow.UpdatePhones` or `CreditCardWindow.UpdateData`, except that the method called by the Check is defined by `PhoneWindow` or `CreditCardWindow`, respectively. These methods are implemented slightly differently, but they're all named `ReportDebugging`, take a single Boolean argument, and return no value.

Action: To make the app close if a debugger is attached, I set the Action property to `Exit`. This tells Dotfuscator to inject code that closes the app when the Check detects an unauthorized state. Note that the Check performs this Action after notifying the application, so in this scenario the incident report would be sent before the app closes.

Configuring the Tamper Check

Finally, I added a Tamper Check to address the reverse-engineering scenario. I clicked the Add Tamper Check button to configure a new Tamper Check, as shown in **Figure 5**.

Locations: Just as with the Query Debugging Check, I chose to give the Tamper Check multiple locations: `LoginDialog.ConfirmLogin`, `CustomerWindow.UpdateData` and `Utilities.ShowAndHandleDialog`.

With a Debugging Check, having multiple locations is important because the debugger could be attached at any time during execution. But a Tamper Check will have only one result over a run of the app—the runtime loaded either an assembly that was modified or not. Shouldn't one location be sufficient? Actually, because this Check is to deter tampered binaries, I have to consider a scenario where an attacker was able to remove the Tamper Check itself from a location. By having multiple locations, the application is more resilient to tampering.

You might notice that one of the locations, `LoginDialog.ConfirmLogin`, is the same as the one for the Login Debugging Check. Dotfuscator allows multiple Checks of different types to be injected at the same location. In this case, after the user logs in, both debugging and tampering will be checked.

Application Notification: For the Application Notification Sink, I decided it would be better in this case to have just one Sink for all of the locations. This is because unlike with the Query Debugging Check, I don't really care on the reporting side which location triggers the Check.

I chose to define the `Utilities.ReportTampering` method as the Sink. As the context of each location varies, I had to declare the Sink static and ensure it was accessible from each location. The method is defined as such:

```
// In Utilities static class
internal static void ReportTampering(bool isTampered)
{
    if (isTampered)
    {
        ClientAppInsights.TelemetryClient.TrackEvent("Tampering Detected");
        ClientAppInsights.Shutdown();
    }
}
```

Whenever any of the Check's locations are called, the Check determines whether it has been modified since Dotfuscator processed it, and then calls the `ReportTampering` method with a parameter of true if modification was detected and false otherwise.

Action: If the app is modified, it's dangerous to continue. I configured this Check's Action to `Exit`, so that the app closes when tampering is discovered.

Injecting the Checks

With the Checks configured, Dotfuscator can now inject them into the app. To do this, from Dotfuscator CE, open the `AdventureWorksSalesClient\Dotfuscator.xml` config file and then click the Build button.

Dotfuscator will process the Client's assembly, inject the Checks, and write the protected Client out to `AdventureWorksSalesClient\Dotfuscated\Release`. The unprotected app remains in `AdventureWorksSalesClient\bin\Release`.

Testing the Checks

As with any security control, it's important to test the app's behavior when the control is introduced.

Normal Scenarios: The Checks shouldn't have any impact on legitimate users of the app. I ran the protected Client normally and saw no unexpected crashes, application exits or Application Insights events.

Unauthorized Scenarios: You should also verify that Checks do what you expect when the app is used in an unauthorized way. The sample code's README lists detailed instructions for testing the Debugging Checks and the Tamper Check. For instance, to test the Query Debugging Check, I ran the protected Client several times, attaching WinDbg to the process at various points. The app correctly reported and responded to the presence of a debugger per the Check's configuration.

Layered Protection Strategy

Using just one protection measure is insufficient for most practical cases, and Checks are no exception. Checks should be just one layer of your protection strategy, along with techniques like end-to-end encryption, Authenticode assembly signing, and so on. When you use multiple protection layers, the strengths of one layer can offset a weakness of another layer.

In this article's example, some Application Notification Sinks used by the Checks have names like `isDebugged` or `ReportTampering`. These names remain in compiled .NET assemblies, and an attacker could easily understand the intent of these code elements and work around them. To mitigate this, Dotfuscator can, in addition to injecting Checks, also perform renaming obfuscation on your assemblies. For details, see the PreEmptive Protection – Dotfuscator Community Edition documentation (bit.ly/2y9oxYX).

Wrapping Up

This article introduced Runtime Checks and some of the problems they solve. Using a LOB app, I demonstrated how a data breach can occur in client software and how Runtime Checks can be configured to detect, report and respond to such a breach.

While this article covered the free Dotfuscator Community Edition, these same concepts can be transferred to the commercially licensed Professional Edition, which has additional features for Checks (bit.ly/2xgEZcs). You can also inject Checks into Java and Android apps with Dotfuscator's sibling product, *PreEmptive Protection - DashO* (bit.ly/2ffHTrN). ■

JOE SEWELL is a software engineer and technical writer on the Dotfuscator team at PreEmptive Solutions.

THANKS to the following Microsoft technical expert for reviewing this article:
Dustin Campbell

Modern Apps **LIVE!**

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

**ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO
November 12-17**

Presented in
Partnership with **Magenic**

Delivering Modern Apps

Presented in partnership with Magenic, Modern Apps Live! brings Development Managers, Software Architects and Development Leads together to break down the complex landscape of mobile, cross-platform, and cloud development and learn how to architect, design and build a complete Modern Application from start to finish.

In-depth and educational sessions taught by the industry's top thought leaders will lay out how to get an app done successfully and at a low cost!



**REGISTER
NOW**

REGISTER BY NOV 12 AND SAVE \$300!*

Must use discount code LEB01

*Savings based on 5-day packages only. See website for details.

A Part of Live! 360: The Ultimate Education Destination

5 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio **LIVE!**

SQL Server **LIVE!**

TECHMENTOR

Office &
SharePoint **LIVE!**

Modern Apps **LIVE!**



MODERNAPPSLIVE.COM

EVENT PARTNERS



PLATINUM SPONSORS



SILVER SPONSOR



SUPPORTED BY



PRODUCED BY



2017
Orlando

ROYAL PACIFIC RESORT
AT UNIVERSAL ORLANDO
NOVEMBER 12-17

Visual Studio **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Coding in Paradise

Grab your flip flops, and your laptops, and make plans to attend Visual Studio Live! (VSLive!™), the conference more developers rely on to expand their .NET skills and the ability to build better applications.

Over six full days of unbiased and cutting-edge education on the Microsoft Platform, developers, engineers, designers, programmers and more will soak in the knowledge on everything from Visual Studio and the .NET framework, to AngularJS, ASP.NET and Xamarin.



CONNECT WITH LIVE! 360



twitter.com/live360
[@live360](https://twitter.com/live360)



facebook.com
Search "Live 360"



linkedin.com
Join the "Live! 360" group!

EVENT PARTNERS



PLATINUM SPONSORS



SILVER SPONSOR



SUPPORTED BY





TECH EVENTS WITH PERSPECTIVE

5 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to four (4) other co-located events at no additional cost:



Five (5) events and hundreds of sessions to choose from—mix and match sessions to create your own, custom event line-up—it's like no other conference available today!

TURN THE PAGE FOR MORE EVENT DETAILS →



NEW: HANDS-ON LABS



Join us for full-day,
pre-conference hands-on labs
Sunday, November 12.

Only \$695 Through November 12

Whether you are an

- Engineer
- Developer
- Programmer
- Software Architect
- Software Designer

You will walk away from this event having expanded your .NET skills and the ability to build better applications.

**REGISTER
NOW**

**REGISTER BY
NOVEMBER 12
& SAVE \$300!***

Must use discount code LEB01

*Savings based on 5-day packages only.
See website for details.

PRODUCED BY

Redmond
HEADQUARTERS

VIRTUALIZATION
& CLOUD REVIEW

Visual Studio
MAGAZINE

I105MEDIA
YOUR GROWTH, OUR BUSINESS.



VSLIVE.COM/ORLANDOMSDN

ALM / DEVOPS		CLOUD COMPUTING	NATIVE CLIENT	SOFTWARE PRACTICES	VISUAL STUDIO / .NET FRAMEWORK	WEB CLIENT
<div>NEW</div> Full Day Hands-On Labs: Sunday, November 12, 2017						
START TIME	END TIME	VSS01 Full Day Hands-On Lab: Busy Developer's HOL on Angular - Ted Neward				VSS02 Full Day Hands-On Lab: From 0-60 in a day with
START TIME	END TIME	Pre-Conference Workshops: Monday, November 13, 2017				
8:30 AM	5:30 PM	VSM01 Workshop: Distributed Cross-Platform Application Architecture - Jason Bock & Rockford Lhotka		VSM02 Workshop: Add Intelligence to Your Solutions with AI, Bots, and More - Brian Randell		VSM03 Workshop: Designing, Developing,
6:30 PM	8:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:30pm - Meet at Conference Registration Desk to walk over with the group				
START TIME	END TIME	Day 1: Tuesday, November 14, 2017				
8:00 AM	9:00 AM	Visual Studio Live! KEYNOTE: Docker, JavaScript and Azure - John Papa, Principal Developer Advocate, Microsoft				
9:15 AM	10:30 AM	VST01 Front-end Web Development in 2017 for the Front-endally Challenged Microsoft Developer - Chris Klug	VST02 Go Mobile with C#, Visual Studio, and Xamarin - James Montemagno		VST03 Microservices with Azure Container Service & Service Fabric - Vishwas Lele	
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - Pacifica 7				
11:00 AM	12:00 PM	LIVE! 360 KEYNOTE: Lessons Learned Doing DevOps at Microsoft at Scale - Buck Hodges, Director of Engineering, Visual Studio Team				
12:00 PM	12:45 PM	Lunch • Visit the EXPO				
12:45 PM	1:30 PM	Dessert Break • Visit the EXPO				
1:30 PM	2:45 PM	VST05 ASP.NET Core MVC—What You Need to Know - Philip Japikse	VST06 Optimizing and Extending Xamarin.Forms Mobile Apps - James Montemagno		VST07 Tactical DevOps with Visual Studio Team Services - Brian Randell	
2:45 PM	3:15 PM	Networking Break • Visit the EXPO - Pacifica 7				
3:15 PM	4:30 PM	VST09 Angular(2)—The 75-Minute Crash Course - Chris Klug	VST10 Building a Cross-Platform Mobile App Backend in the Cloud - Nick Landry		VST11 Database Lifecycle Management and the SQL Server Database - Brian Randell	
4:40 PM	5:00 PM	VST13 Fast Focus: Aurelia vs. Just Angular - Chris Klug	VST14 Fast Focus: Tips & Tricks for Xamarin Development - James Montemagno		VST15 Fast Focus on Azure Functions - Brian Clark	
5:10 PM	5:30 PM	VST17 Fast Focus: Web Security 101 - Brock Allen	VST18 Fast Focus: Cross-Platform Code Reuse - Rockford Lhotka		VST19 Fast Focus: Exploring Microservices in a Microsoft Landscape - Marcel de Vries	
5:30 PM	7:30 PM	Exhibitor Reception - Pacifica 7				
START TIME	END TIME	Day 2: Wednesday, November 15, 2017				
8:00 AM	9:15 AM	VSW01 User Authentication for ASP.NET Core MVC Applications - Brock Allen	VSW02 Cloud Oriented Programming - Vishwas Lele		VSW03 Overcoming the Challenges of Mobile Development in the Enterprise - Roy Cornelissen	
9:30 AM	10:45 AM	VSW05 Building AngularJS Component-Based Applications - Miguel Castro	VSW06 Building Modern Web Apps with Azure - Eric D. Boyd		VSW07 Designing Beautiful REST+JSON APIs with Ion- Les Hazlewood	
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - Pacifica 7				
11:30 AM	12:30 PM	LIVE! 360 KEYNOTE: Live! 360 Panel: Industry Trends, Technology, and Your Career - Matt Lockhart, Michael Aaron, Martin Leifker,				
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch • Visit the EXPO				
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO				
2:00 PM	3:15 PM	VSW09 Securing Web APIs in ASP.NET Core - Brock Allen	VSW10 A/B Testing, Canary Releases and Dark Launching, Implementing Continuous Delivery on Azure - Marcel de Vries		VSW11 Creating a Release Pipeline with Team Services - Esteban Garcia	
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - Pacifica 7				
4:00 PM	5:15 PM	VSW13 Build Object-Oriented Enterprise Apps in JavaScript with TypeScript - John Papa	VSW14 Lock the Doors, Secure the Valuables, and Set the Alarm - Eric D. Boyd		VSW15 Unit Testing & Test-Driven Development (TDD) for Mere Mortals - Benjamin Day	
8:00 PM	10:00 PM	Live! 360 Dessert Luau - Wantilan Pavilion				
START TIME	END TIME	Day 3: Thursday, November 16, 2017				
8:00 AM	9:15 AM	VSH01 HTTP/2: What You Need to Know - Robert Boedigheimer	VSH02 PowerApps and Flow Part II: Package, Embed, and Extend Your Applications - Manas Maheshwari & Pratap Ladhani		VSH03 Exploring C# 7 New Features - Adam Tuliper	
9:30 AM	10:45 AM	VSH05 ASP.NET Tag Helpers - Robert Boedigheimer	VSH06 Storyboarding 101 - Billy Hollis		VSH07 .NET Standard—From Noob to Ninja - Adam Tuliper	
11:00 AM	12:00 PM	Visual Studio Live! Panel: Native or Web—Does it Matter? - Brian Randell (Moderator), Damian Brady, Jeremy Clark, Esteban Garcia, Billy Hollis, & Adam Tuliper				
12:00 PM	1:00 PM	Lunch on the Lanai - Lanai / Pacifica 7				
1:00 PM	2:15 PM	VSH09 I See You: Watching the User with Reactive Forms - Deborah Kurata	VSH10 Continuous Integration and Deployment for Mobile Using Azure Services - Kevin Ford		VSH11 Deploying Straight to Production: A Guide to the Holy Grail - Damian Brady	
2:30 PM	3:45 PM	VSH13 Angular Routing - Deborah Kurata	VSH14 XAML Inception—Deep Composition for Better UI - Billy Hollis		VSH15 Application Insights: Measure Your Way to Success - Esteban Garcia	
4:00 PM	5:00 PM	Next? Visual Studio Live! Networking Event - Brian Randell (Moderator), Damian Brady, Jeremy Clark, Esteban Garcia, Billy Hollis, & Deborah Kurata				
START TIME	END TIME	Post-Conference Workshops: Friday, November 17, 2017				
8:00 AM	5:00 PM	VSF01 Workshop: Angular Fundamentals - John Papa			VSF02 Workshop: Building, Running & Microservices with Docker Containers on Azure	

WEB SERVER

MODERN APPS LIVE!

Check Out These Additional Sessions for Developers at Live! 360

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live! features 15+ developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Developing Extensions for Microsoft Teams - *Paul Schaefflein*
- Workshop: Mastering the SharePoint Framework - *Andrew Connell*
- TypeScript for SharePoint Developers - *Rob Windsor*
- Building Office Add-ins for Outlook with Angular - *Andrew Connell*
- Developing SharePoint Framework Components Using Visual Studio - *Paul Schaefflein*
- What Every Developer Needs to Know about SharePoint Development Online or On-Prem - *Robert Bogue*
- Build a Complete Business Solution Using Microsoft Graph API through Client Side Web Parts - *Julie Turner*



SQL Server LIVE!

TRAINING FOR DBAS AND IT PROS

SQL Server Live! features 30+ developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Developer Dive into SQL Server 2016 - *Leonard Lobel*
- Turbo Boost - SQL Tricks Everybody MUST Know - *Pinal Dave*
- Advanced SSIS Package Authoring with Biml - *Tim Mitchell*
- Graph DB Support in SQL Server 2017 - *Karen Lopez*
- Big Data Technologies: What, Where and How to Run Them on Azure - *Andrew Brust*
- Top Five SQL Server Query Tuning Tips - *Janis Griffin*
- Workshop: Big Data, BI, and Analytics on The Microsoft Stack - *Andrew Brust*



TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

TechMentor features 20+ developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Ethical Hacking with Kali Linux - *Mike Danseglio & Avril Salter*
- Workshop: Windows Security—How I Do It! - *Sami Laiho*
- Hardware, Camtasia, and a Storyline: Creating Your Own User Training - *Greg Shields*
- Make Your PowerShell Scripts Bulletproof with Pester - *Melissa Januszko*
- Controlling Your Azure Spend - *Timothy Warner*
- PowerShell Scripting Secrets - *Jeffery Hicks*
- In-Depth Introduction to Docker - *Neil Peterson*



VSLIVE.COM/ORLANDOMSDN

Xamarin and Xamarin.Forms - *Roy Cornelissen*

Service Oriented Technologies: & Implementing WCF and the Web API - *Miguel Castro*

VST04 What's New in Visual Studio 2017 - *Robert Green*

Services, Microsoft - *Pacifica 6*

VST08 Building ASP.NET Apps on Google Cloud - *Mete Atamel*

VST12 Top 10 Entity Framework Core Features Every Developer Should Know - *Philip Japikse*

VST16 Fast Focus: Busting .NET Myths - *Jason Bock*

VST20 Fast Focus: Dependency Injection in 20 Minutes - *Miguel Castro*

VSW04 Building Apps with Microsoft Graph and Visual Studio - *Robert Green*

VSW08 Bots are the New Apps: Building Bots with ASP.NET Web API & Language Understanding - *Nick Landry*

Francois Charette - *Pacifica 6*

VSW12 Azure Functions Webhook Interface - *Nick Zimmerman*

VSW16 PowerApps, Flow, and Common Data Service: Empowering Businesses with the Microsoft Business Application Platform - *Charles Sterling*

VSH04 Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*

VSH08 Devs vs. Ops: Making Friends with the Enemy - *Damian Brady*

VSH12 Design Patterns: Not Just for Architects - *Jeremy Clark*

VSH16 DI Why? Getting a Grip on Dependency Injection - *Jeremy Clark*

Continuously Deploying - *Marcel de Vries & Rene van Osnabrugge*

Pre-Con Workshops: Monday, Nov. 13

MAM01 Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*

Dine-A-Round Dinner

Day 1: Tuesday, November 14, 2017

Modern Apps Live! KEYNOTE: Docker, JavaScript and Azure - *John Papa, Principal Developer Advocate, Microsoft*

MAT01 Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*

Networking Break • Visit the EXPO - *Pacifica 7*

LIVE! 360 KEYNOTE

Lunch • Visit the EXPO

Dessert Break • Visit the EXPO

MAT02 Architecture: The Key to Modern App Success - *Brent Edwards*

Networking Break • Visit the EXPO - *Pacifica 7*

MAT03 Focus on the User Experience #FTW - *Jim Barrett*

MAT04 Fast Focus: Hybrid Web Frameworks - *Allen Conway*

MAT05 Fast Focus: Web Assembly - *Jason Bock*

Exhibitor Reception - *Pacifica 7*

Day 2: Wednesday, November 15, 2017

MAW01 Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randall*

MAW02 DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*

Networking Break • Visit the EXPO - *Pacifica 7*

LIVE! 360 KEYNOTE

Birds-of-a-Feather Lunch

Dessert Break • Visit the EXPO

MAW03 Security with Speed for Modern Developers - *Michael Lester*

Networking Break • Visit the EXPO • Raffle @ 3:30 p.m.

MAW04 Coding for Quality and Maintainability - *Jason Bock*

Live! 360 Dessert Luau - *Wantilan Pavilion*

Day 3: Thursday, November 16, 2017

MAH01 Modern Web Development: Building Server Side Using ASP.NET Core, MVC, Web API, and Azure - *Allen Conway*

MAH02 Modern Web Development: Building Client Side Using TypeScript and Angular - *Allen Conway*

Modern Apps Live! Panel: Mobile Development Technologies - *Rockford Lhotka (Moderator), James Montemagno, Kevin Ford*

Lunch on the Lanai - *Lanai / Pacifica 7*

MAH03 Using All That Data: Power BI to the Rescue - *Scott Diehl*

MAH04 Modern Mobile Development: Build a Single App For iOS, Android, and Windows with Xamarin Forms - *Kevin Ford*

Next? Modern Apps Live! Networking Event - *Rockford Lhotka (Moderator)*

Post-Con Workshops: Friday, Nov. 17

MAF01 Workshop: Modern App Deep Dive—Xamarin, Responsive Web, UWP - *Kevin Ford, Brent Edwards, Allen Conway*

Continuous Data Migration Using Visual Studio and TFS

Jebarson Jebamony

Data migration is often a requirement in an application development effort, whether it's a greenfield project or an application redesign. Despite this, very little attention gets paid to data migration during the design and development phase, with activity often pushed back to the latter part of the project. While this approach might allow an undivided focus on migration, it comes with a few risks, not the least of which is that teams lack the time and resources to properly test the migration and surrounding code.

The concept of continuous data migration is founded on the idea of developing migration scripts during the development of an application and maintaining them through versions just like the application code. A continuous migration approach allows you to test your migrations alongside code development, ensuring your data and code assets stay in sync.

This article discusses:

- Introduction of continuous data migration as a process to software development
- Implementing versioning in a database
- Implementing continuous data migration in SQL using Visual Studio and Team Foundation Server

Technologies discussed:

Visual Studio, Team Foundation Server, SQL Server

Code download available at:

github.com/Jebarson/ContinuousDataMigration

In this article, I describe a solution that leverages Visual Studio and Team Foundation Server to achieve continuous data migration. Keep in mind there are third-party tools like Red Gate Ready Roll that are capable of doing this partially, but they come at a huge cost and lack the capability of continuous data migration.

Challenge and Solution

Visual Studio can perform an incremental publish of a database with the help of `SqlPackage.exe`, but the tool is lacking in many ways. For instance, `SqlPackage.exe` falls short when inserting a new column between the columns of a table, changing the seed data, and normalizing and de-normalizing tables, among other examples.

Also, versioning changes is very important when you need to do targeted fixes and deployment. For example, you may need to increment the value of a column by 10 when you migrate from v1.2 to v1.3, but not in any other flow. This can only be achieved with the use of versioning; however, SQL lacks this capability.

I want to approach the challenge by designing a solution that takes full advantage of what Visual Studio and `SqlPackage.exe` can offer, while resolving the previously mentioned shortfalls.

A typical database project has two types of scripts—compiled and non-compiled. All of the objects such as schema, tables, views, stored procedures and the like are generally written as a compiled script. The seed script and any runtime queries will generally be placed into the post-deployment script, which is non-compiled.

Let's start with an example. **Figure 1** shows the AdventureWorks.Database sample database project (imported from the

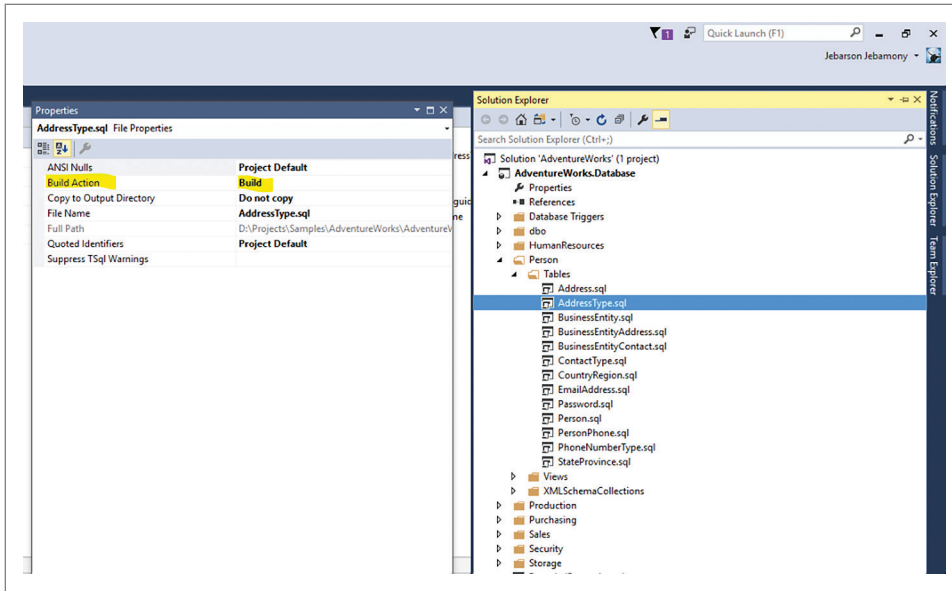


Figure 1 Compiled Scripts in the AdventureWorks.Database

backup available at bit.ly/2vPwu4N). As you can see, all the objects are put in as compiled scripts.

The seed data script (containing data needed for the application to function) is put in as non-compiled script and referred in the post-deployment script. **Figure 2** shows this. If you are not aware of the significance of post-deployment scripts, I encourage you to read the MSDN Library documentation at bit.ly/2w12ly4.

To ensure that the post-deployment script can handle incremental deployment, I've added a NOT EXISTS clause in front of all the INSERT statements. For example:

```
IF NOT EXISTS (SELECT 1 FROM [Person].[AddressType] WHERE [AddressTypeID] = 1)
INSERT [Person].[AddressType] ([AddressTypeID], [Name], [ModifiedDate])
VALUES (1 N'Billing', CAST (N'2008-04-30T00:00.000' AS DateTime))
```

For the sake of simplicity and ease of maintenance, I'll keep all the seed scripts on their respective files and refer them back in the post-deployment script.

I now have a project that will deploy the latest schema and seed data, at any point in time. It is also capable of performing an

incremental deployment on an existing database if the project is not introducing any breaking changes. However, the restrictions I mentioned at the start of this section come into play.

Finally, there is a bug that breaks the incremental deployment when a user-defined type (UDT) is changed. Unfortunately, the Visual Studio team has marked this bug as won't fix, which means you'll need to work around that. You can explore more details about the bug in the Visual Studio Developer Community entry at bit.ly/2w0zTBU.

Versioning

Just as you version any application you ship, it's important to version the database, as well. Versioning

helps you keep track of the source code so you can easily keep tabs on the features, bugs and fixes made in every release of the software. If you're not already familiar with versioning, take a moment and check out the article, "Semantic Versioning 2.0.0" at semver.org. It's worth a read.

Before I get started, I have to address a challenge: SQL doesn't actually come with a mechanism for versioning, so I'll need to create one of my own. I'll create a table called [internal].[DatabaseVersion] to store the version detail, where "internal" is the schema of the table. It's good practice to have a separate schema for all the database objects used for internal purposes (that is, they don't participate in the actual business).

Figure 3 shows the schema I would propose for the table. You can follow your own pattern if you're not convinced. Just keep in mind that we create versions to keep track of builds and releases.

Every time I make a change to the schema or check in a data migration script, I add a new version entry to the table, which serves

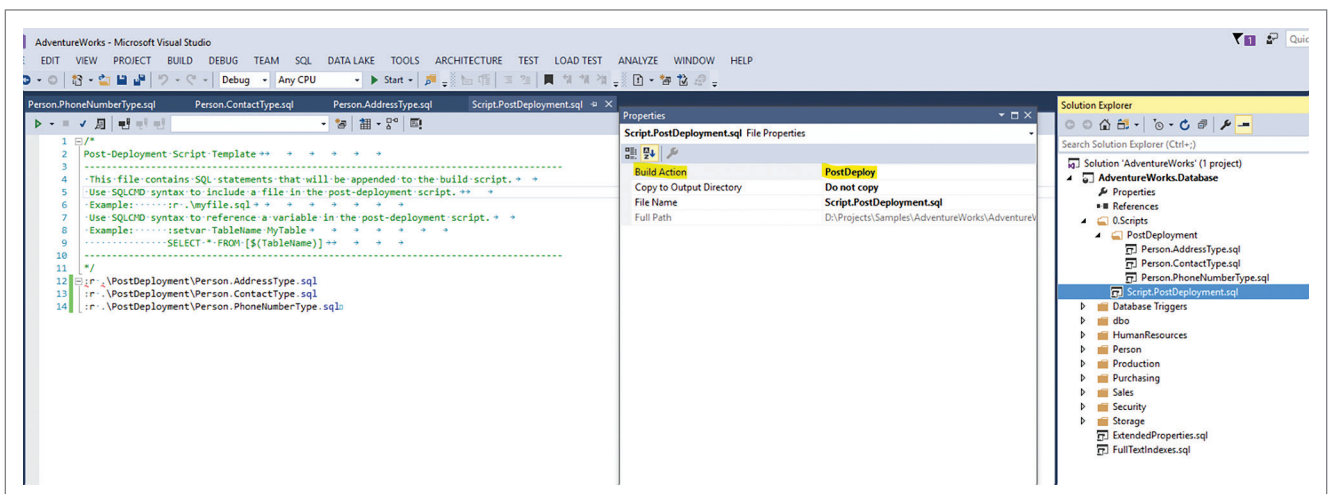


Figure 2 Post-Deployment Scripts

Figure 3 Table Schema

```
CREATE TABLE [internal].[DatabaseVersion]
(
    [DatabaseVersionId] INT IDENTITY(1,1) NOT NULL,
    [Major] INT NOT NULL,
    [Minor] INT NOT NULL,
    [Build] INT NOT NULL,
    [Revision] INT NOT NULL,
    [CreatedBy] NVARCHAR (256)
    CONSTRAINT [DFDatabaseVersionCreatedBy] DEFAULT (') NOT NULL,
    [CreatedOn] DATETIME
    CONSTRAINT [DFDatabaseVersionCreatedOn] DEFAULT (GETUTCDATE()) NOT NULL,
    [ModifiedBy] NVARCHAR (256)
    CONSTRAINT [DFDatabaseVersionModifiedBy] DEFAULT (') NOT NULL,
    [ModifiedOn] DATETIME
    CONSTRAINT [DFDatabaseVersionModifiedOn] DEFAULT (GETUTCDATE()) NOT NULL,
    CONSTRAINT [PKDatabaseVersion] PRIMARY KEY CLUSTERED ([DatabaseVersionId] ASC)
);GO
```

as the label for the change. If the current version is 1.0.0.0, and I introduce a migration that fixes the Gender flag issue by reversing the values, I will add appropriate scripts to make that change and add a new entry to the table with the version saying 1.1.0128.212.

Migration

As I already discussed, Visual Studio can do incremental deployments, but not with breaking changes. So as I design the migration, I need to keep that in mind and work around the limitation.

The first step is to create a separate project to target the migration. With respect to the sample in **Figure 3**, I create a new database project called AdventureWorks.Database.Migration. This migration project targets two types of scripts. The first is the data migration script, which needs to be run if any data movement or update occurs. The second script takes care of breaking schema changes that Visual Studio and SqlPackage.exe cannot handle. Both these scripts go into the project as a post-deployment script. There are no compilable scripts in this project.

To better understand the scenario, let's discuss everything in relation to the AdventureWorks sample. I've uploaded the source code for this into my GIT repository at github.com/Jeberson/ContinuousDataMigration. The master branch incorporates the base project that I've imported and created from the database, as mentioned earlier.

Before I venture into the scenario, I'd like to explain how the migration works. As I discussed in the Versioning section, I'm versioning every single change released by adding a new row into internal.DatabaseVersion. Inside the AdventureWorks.Database.Migration project, I write the logic to execute appropriate migration

scripts based on the targeted database version. Check out the flowchart in **Figure 4** for an understanding of the logic involved.

At the start of the AdventureWorks.Database.Migration project, I check the current version of the database and based on that run the migration scripts all the way to the latest version. Here's the code snippet I use to decide the migration path, which I'll refer to as Script 1:

```
DECLARE @currentDBVersion BIGINT = NULL;

-- Get the current version of the database.
SELECT TOP 1 @currentDBVersion = Id FROM [internal].[DatabaseVersion]
ORDER BY [DatabaseVersionId] DESC

-- Jump to the incremental migration scripts based on the current version.
IF @currentDBVersion = 1 GOTO Version11
ELSE IF @currentDBVersion = 2 GOTO Version12
ELSE
    RETURN
```

Now that I've shown how the migration scripts run, let's drive the migration with a few fictional scenarios that help illustrate what's going on. I'll discuss two version changes from the base project that I created earlier.

Version 1.1: This is the first change on top of the base project I created. The changes are available in the v11 branch of the Continuous Data Migration project on GitHub. The changes I've committed in this version are as follows:

- Inserted a new IsEmployee column to [HumanResources].[Employee] after the JobTitle column.
- Changed the [Person].[AddressType] name from Main Office to Office.
- Changed SPs (no need to include in migration project).
- New SPs (no need to include in migration project).

All these changes are made into the regular project AdventureWorks.Database as is, along with the new version row into internal.DatabaseVersion. This facilitates any fresh deployment to incorporate the latest changes. For any existing database with the base version to be upgraded to v1.1, I need to implement the same changes into the migration project. To do so, I segregate it into two sections: the schema change and the data change. Inserting a new column called IsEmployee is a schema change, while changing AddressType from Main Office to Office is a data change.

The schema change is something Visual Studio can do. However, it can only append the column, and that's something I don't want. To overcome this limitation, I must generate a script to first drop all the dependencies (indexes, constraints, foreign keys and the like) of the table Employee, and then create a temporary table with the new column in the correct order with all the dependencies.

Then, I can move the data from the Employee table to the temp table, drop the Employee table and finally rename the temp table to Employee. This script is available in the v11 branch of my Continuous Data Migration project on GitHub, in the file SchemaChangeScript.sql.

The data change just alters a value of the record from Main Office to Office and, therefore, I can script an update query to accomplish this. Check out the

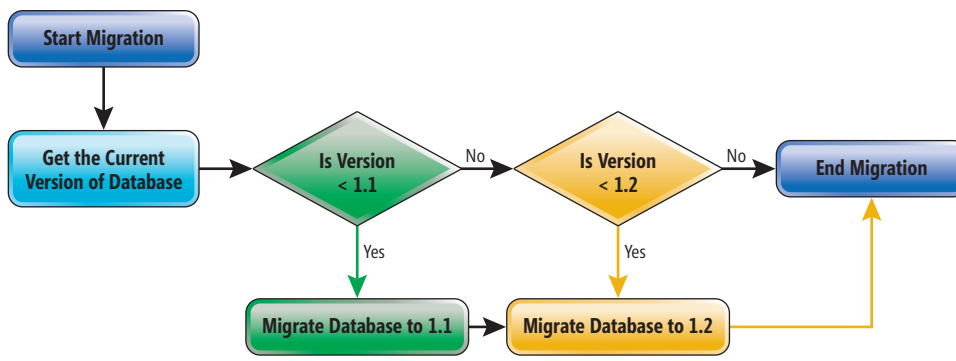


Figure 4 The Migration Logic

Spreadsheets Made Easy.



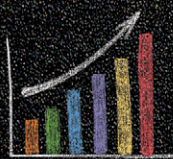
SpreadsheetGear 2017 Released

SpreadsheetGear 2017 adds a new SpreadsheetGear for .NET Standard product, official support for Excel 2013 and Excel 2016, 51 new Excel functions for a total of 449 fully supported functions, full conditional formatting support, enhanced workbook protection and encryption, cell gradient rendering and more.



Support for iOS, Android, Linux, macOS, UWP and more

SpreadsheetGear for .NET Standard enables cross-platform developers to enjoy the same high performance Excel-compatible reporting, charting, calculations and more relied on by thousands of Windows developers for 10+ years.



Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.

Download your free fully functional evaluation at SpreadsheetGear.com



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.



Windows
Forms



Silverlight



WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.



Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.



SpreadsheetGear

Figure 5 Migration Project Cheat List

Change	Triage
New table/view/stored procedure/object	Leverage Visual Studio
Change in view/stored procedure/function	Leverage Visual Studio
Change in user-defined type	Drop all related stored procedures of the UDT. This is a workaround for the bug described earlier.
Addition of new column to table	Script a migration from the existing table to a new table with the correct column order (see github.com/Jebbarson/ContinuousDataMigration). This isn't required if you're adding a nullable column and the order of the column is immaterial.
Normalization or de-normalization of table	Script a migration to either split or merge based on the requirement. This is similar to the script created in v1.2.
Change in data	Script out the data change.

DataChangeScript.sql file in the v11 branch of the Continuous Data Migration project on GitHub.

When the migration project is run on top of the existing AdventureWorks.Database, the code from Script 1 will send the execution to a script that calls the schema and data change script, as shown in the following snippet, which I'll call Script 2:

```
-- Script to migrate to v1.1
Version11:
:r .\Scripts\Migration\V11\SchemaChangeScript.sql
:r .\Scripts\Migration\V11\DataChangeScript.sql

EXEC [internal].[CreateDatabaseVersion] @id = 2, @major = 1, @minor = 1,
    @build = 0128,
    @revision = 212
```

Version 1.2: This is the latest change committed on top of v1.1. The same changes are available in the v12 branch of the project on GitHub. The changes in this version are as follows:

- Changed the IsEmployee in [HumanResources].[Employee] to EmployeeType, referring a new table to [HumanResources].[EmployeeType].
- Changed SPs (no need to include in migration project).
- New table (no need to include in migration project).

Similar to v1.1, I've also made the changes on the regular project AdventureWorks.Database along with a new entry into internal.DatabaseVersion. As you can see, IsEmployee is now changed to EmployeeType to accommodate more types of employees. To achieve this, I follow the same pattern I did in v1.1. However, I need to write the data migration for the new column based on the value of the earlier column. The schema change script is written in the file SchemaChangeScript.sql in the v12 branch of the Continuous Data Migration project on GitHub.

Here's the script I've included in the project to migrate to v1.2, which I'll call Script 3:

```
-- Script to migrate to v1.2
Version12:
:r .\Scripts\Migration\V12\SchemaChangeScript.sql

EXEC [internal].[CreateDatabaseVersion] @id = 3, @major = 1, @minor = 2,
    @build = 0414,
    @revision = 096
```

As I mentioned earlier, Visual Studio is partially capable of driving an incremental deployment, but I'm targeting only the items that Visual Studio lacks with the scripts I've generated up to this point. You may have noticed that for certain items in both v1.1 and v1.2, I

mentioned "no need to include in migration project." That's because Visual Studio is able to deploy them incrementally. Which begs the question: What changes qualify for a migration project and what changes don't?

You can look at the handy cheat list in **Figure 5**, which will help you decide whether to script the migration or not. Note that there might be more items you come across that could be added to this list.

OK, enough about generating the migration scripts. Time to move on to the deployment.

When deploying a fresh instance of the latest version of an existing database, there's no need for migration. In the example I've been presenting, all you need to deploy is the

AdventureWorks.Database. You can do this from either Visual Studio (via publish) or using SqlPackage.exe. Here's the command for deploying the database using SqlPackage.exe:

```
SqlPackage.exe /Action:Publish /SourceFile:"AdventureWorks.Database.dacpac" /tdn:<<DatabaseName>> /tsn:"<<SQL Instance>>"
```

3 Common Migration Problems and Fixes

Continuous data migration can produce a lot of benefits, but it's not without its challenges. Here are some common issues you might expect to encounter while implementing this solution, and ways to address them.

Build server throws the invalid object name runtime error:

You might encounter this error in the migration project, when your recent version of a migration script has removed an object, but the object is referred in an earlier version script. The resolution is to write queries as sp_executesql '<<your migration script here>>'. For example:

```
EXEC sp_executesql 'ALTER TABLE Employee ADD NewCol INT'
```

Out of control migration scripts and version overload:

It's a good idea to always set a minimum target version for migration. Doing so limits the scope of your migration scripts and helps ensure that they don't become too difficult to maintain.

Implementing with a production database:

In case you want to implement this solution in a database that's already in production, include the definition of internal.DatabaseVersion and its version entries. Change the "Script 1" to see if the table internal.DatabaseVersion exists and if it doesn't, direct the execution to the newer version label, which will do the migration and also create the table. For example:

```
DECLARE @currentDBVersion BIGINT = NULL;

-- Get the current version of the database.
IF NOT EXISTS(SELECT 1 FROM [INFORMATION_SCHEMA].[TABLES] WHERE [TABLES].[TABLE_NAME]='DatabaseVersion' AND [TABLES].[TABLE_SCHEMA]='internal')
    SELECT @currentDBVersion = 1
ELSE
    SELECT TOP 1 @currentDBVersion = Id FROM [internal].[DatabaseVersion]
    ORDER BY [DatabaseVersionId] DESC

-- Jump to the incremental migration scripts based on the current version.
IF @currentDBVersion = 1 GOTO Version11
ELSE IF @currentDBVersion = 2 GOTO Version12
ELSE
    RETURN
```

Configuring the TFS Build to Deploy Continuous Migration

The goal is to automate migrations just like continuous integration, so that the build server does the data migration and makes it available to developers and testers once the build is triggered. The next step is to configure the build's release tasks.

To create tasks for the build, you should first be aware of creating a Continuous Integration build. If you aren't, be sure to read through the tutorial posted on the Microsoft Visual Studio site at bit.ly/2xWqtUx.

Once you've created the build tasks, you need to create the deploy tasks for the database. In this example, you must add two deploy tasks: one for the AdventureWorks.Database.Migration project and the other

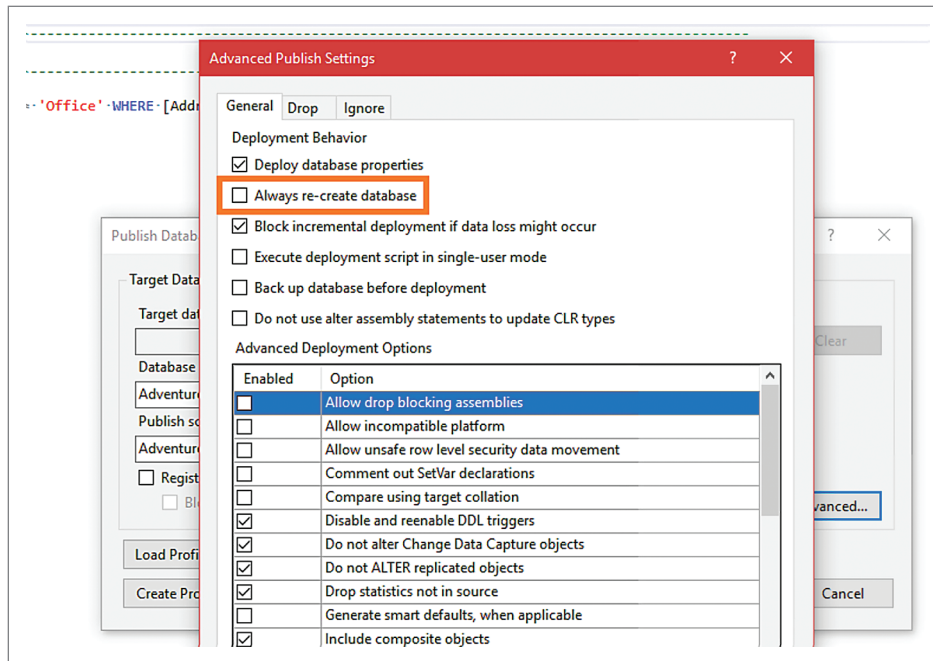


Figure 6 The Advanced Publish Settings Dialog Box

When you perform an incremental deployment over an existing database, there's a chance that the latest script will need a migration. That means I have to deploy the migration database, as well. I'll do this by deploying the AdventureWorks.Database.Migration project first, followed by AdventureWorks.Database. Be sure the "Always re-create database" option is unchecked under the Advanced Deployment Options area of the Advanced Publish Settings dialog box, as shown in **Figure 6**.

If you're running from the command prompt, the command is:

```
SqlPackage.exe /Action:Publish /SourceFile:"AdventureWorks.Migration.Database.dacpac" /tdn:<<DatabaseName>> /tsn:<<SQL Instance>> /p:CreateNewDatabase = False
SqlPackage.exe /Action:Publish /SourceFile:"AdventureWorks.Database.dacpac" /tdn:<<DatabaseName>> /tsn:<<SQL Instance>> /p:CreateNewDatabase = False
```

for the AdventureWorks.Database project. The deploy task will look something like **Figure 7**.

Fill in the detail and set up the trigger based on your requirement. Once the build is up and running you will have set up a Continuous Data Migration for your application.

Wrapping Up

In this article, I explained the importance of continuous data migration on a project that involves multiple phases of release, and how to achieve it using Visual Studio and TFS. Continuous data migration helps you reduce both development effort and migration bugs. In my experience, I've been able to gain as much as 40 percent of migration effort in terms of development. It also got rid of the Migration Phase in the project.

Integration of migration scripts to TFS is as important as the migration scripts themselves. The continuous data migration process is of no use if you aren't deploying it as a part of daily build. "Fail early, fail fast" is the mantra to remember in software development, and continuous data migration sets you up to do exactly that. ■

JEBARSON JEBAMONY is a senior consultant with Microsoft Services and designs and builds solutions for Microsoft, its partners and customers. He has more than 14 years of technical experience and has worked on many Microsoft technologies in that time.

THANKS to the following Microsoft technical experts for reviewing this article: Sagar Dheram and Shrenik Jhaveri

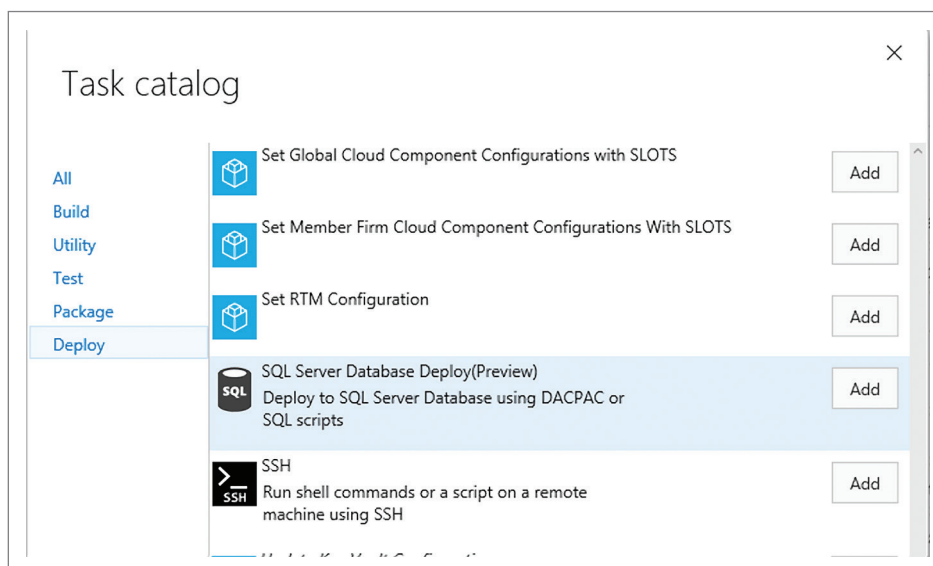


Figure 7 The Deploy Task

Visual Studio[®] LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

March 12 – 16, 2018
Bally's Hotel & Casino
Las Vegas

Respect the Past. Code the Future.

Visual Studio Live! (VSLive!™) Las Vegas, returns to the strip, March 11 – 16, 2018. During this intense week of developer training, you can sharpen your skills in everything from ASP.NET to Xamarin.

Plus, celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Experience the education, knowledge-share and networking at #VSLive25.



VSLive! 1998



VSLive! 2017

SUPPORTED BY



PRODUCED BY



DEVELOPMENT TOPICS INCLUDE:



VS2017/.NET



Angular/JavaScript



ASP.NET Core



Xamarin



Azure / Cloud



Hands-On Labs



Software Practices



ALM / DevOps



SQL Server 2017



UWP (Windows)



Who Should Attend and Why

We've been around since 1993. What's our secret? YOU! Since our first conference (VBITS/ VSLive!/Visual Studio Live!), tens of thousands of developers, software architects, programmers, engineers, designers and more have trusted us year-in-and-year-out for unbiased and cutting-edge education on the Microsoft Platform.



Register to code with us today!

Register by December 15 and Save \$500!

Use promo code VSLNOV2

CONNECT WITH US



twitter.com/vslive –
@VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!

vslive.com/lasvegas



Guidelines for ASP.NET MVC Core Views

Although ASP.NET Core might seem very similar to classic ASP.NET MVC on the surface, there are a lot of differences under the hood. Controllers, Razor views and even model classes can often be easily migrated with minimal effort, yet architecturally speaking, ASP.NET Core diverges significantly from previous non-Core versions of ASP.NET.

The main reason for this is the rewritten pipeline, which provides an ASP.NET Core application at least two ways to generate an HTML-based response. As expected, an application can adopt the MVC programming model and generate HTML out of Razor views invoked via controller actions. Alternately, an application can act as a very thin Web server built around some terminating middleware, and your code in that terminating middleware can do everything, including returning a string that the browser treats as HTML. Finally, in ASP.NET Core 2.0 you can use an entirely new approach—Razor Pages. Razor Pages are Razor views that don't need the MVC infrastructure, but can generate and serve HTML directly out of Razor files.

In this article, after a quick discussion on mini servers, terminating middleware and Razor Pages, I'll contrast and compare the approaches you have within the MVC programming model to build views. In particular, I'll focus on what's new in ASP.NET Core, including tag helpers, view components and dependency injection (DI), and their impact on actual coding practices.

HTML from the Terminating Middleware

The terminating middleware is the final chunk of code in the ASP.NET Core pipeline that gets to process a request. Basically, it's a direct (lambda) function where you process the HTTP request to produce any sort of detectable response, whether plain text, JSON, XML, binary or HTML. Here's a sample Startup class for this purpose:

```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            var html = BuildHtmlFromRequest(context);
            await context.Response.WriteAsync(html);
        });
    }
}
```

By writing HTML-formatted text in the response's output stream, and setting the appropriate MIME type, you can serve HTML content to the browser. It all happens in a very direct way, with no filters and no intermediation, but it definitely works and is faster than anything in earlier versions of ASP.NET. Terminating middleware gives you control over the flow sooner than any other

option. Clearly, coding an HTML factory right in the terminating middleware is far from being a maintainable and flexible solution, but it works.

Razor Pages

In ASP.NET Core 2.0, Razor Pages provide an additional way to serve HTML content, directly invoking a Razor template file without having to go through a controller and action. As long as the Razor page file is located in the Pages folder, and its relative path and name match the requested URL, the view engine will process the content and produce HTML.

What really differentiates Razor pages and Razor views is that a Razor page can be a single file—much like an ASPX page—that contains code and markup. A Razor Page is a CSHTML file flagged with an @page directive that can be bound to a view model that inherits from the system-provided PageModel class. As I mentioned already, all Razor Pages go under the new Pages root project folder and routing follows a simple pattern. The URL is rooted in the Pages folder and the .cshtml extension is stripped off the actual file name. For more information on Razor Pages, have a look at bit.ly/2wp0dUE. In addition, for a more advanced example, have a look at msdn.com/magazine/mt842512.

If you're used to working with MVC controllers, I expect you'll find Razor Pages fundamentally pointless, perhaps only minimally helpful in those rare scenarios where you have a controller method that renders out a view without any business logic around. On the other hand, if you're new to the MVC application model, Razor Pages provide yet another option for mastering the ASP.NET Core framework. To some people, Razor Pages offer a lower barrier of entry for making progress with the framework.

Tag Helpers

The Razor syntax has always been essentially an HTML template interspersed with snippets of C# code. The @ symbol is used to tell the Razor parser where a transition occurs between HTML static content and a code snippet. Any text following the @ symbol is parsed according to the syntax rules of the C# language. The items discovered during text parsing are concatenated to form a dynamically built C# class that's compiled on the fly using the .NET Compiler Platform ("Roslyn"). Executing the C# class just accumulates HTML text in the response stream, mixing together static content and dynamically computed content. In the end, the expressivity of the Razor language is limited to the expressivity of HTML5.

Free AppFabric Wrapper
Quick AppFabric Migration to NCache



Extreme Performance Linear Scalability



Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing



NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

100% Native .NET

Open Source

With Razor, the ASP.NET team also introduced an artifact known as HTML helpers. An HTML helper is a sort of small HTML factory that gets some input data and emits just the HTML you want. However, HTML helpers never won over developers, so in ASP.NET Core a much better tool has been added: tag helpers. Tag helpers play the same role as HTML helpers—they work as HTML factories—but provide a much more concise and natural syntax. In particular, you don't need any C# code to tie tag helpers with the Razor template code. Instead, they look like elements of an extended HTML syntax. Here's an example of a tag helper:

```
<environment names="Development">
  <script src="/content/scripts/yourapp.dev.js" />
</environment>
<environment names="Staging, Production">
  <script src="/content/scripts/yourapp.min.js"
    asp-append-version="true" />
</environment>
```

The environment markup element isn't emitted verbatim to the browser. Instead, its entire subtree is parsed server-side by a tag helper component that's able to read attributes, and inspect and modify the current HTML tree. In this example, the tag helper responsible for the environment element matches the current ASP.NET Core environment to the content of the names attribute, and emits only the script elements that apply. In addition, the script element is processed by another tag helper that inspects the tree looking for an asp-append-version custom attribute. If found, the actual URL being generated is appended with a timestamp to ensure that the linked resource is never cached.

Tag helpers are C# classes inherited conventionally from a base class or declaratively bound to markup elements and attributes. Each Razor file that intends to use tag helpers must declare them using the @addTagHelper directive. The directive simply registers tag helper classes from a given .NET Core assembly. The @addTagHelper directive may appear in individual Razor files but more commonly goes in the _ViewImports.cshtml file and applies globally to all views:

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Tag helpers are a cool piece of technology, but they represent a trade-off between expressivity and cost of server-side rendering.

Attributes and elements recognized as tag helpers are also emphasized in Visual Studio with a special color. ASP.NET Core comes with a full bag of predefined tag helpers that can be grouped in a few categories. Some affect particular HTML elements you can have in a Razor template, such as form, input, textarea, label and select. Some helpers instead exist to simplify and automate the display of validation messages within forms. All predefined tag helpers share the asp-* name prefix. More information can be found at bit.ly/2w3BSS2.

Tag helpers help keep the Razor source code quite readable and concise. In light of this, there would be no reason to avoid using

helpers. Overall, I would recommend using tag helpers to automate the writing of long, repetitive blocks of markup code, rather than creating anything like a view-specific language. The more you use tag helpers, in fact, the more you drive yourself away from plain HTML and the more expensive you make the rendering of HTML views. Tag helpers are a cool piece of technology, but they represent a trade-off between expressivity and cost of server-side rendering. Speaking of benefits, it's also worth mentioning that tag helpers don't alter the overall markup syntax so you can easily open a Razor file with tag helpers in any HTML editor without incurring parsing issues. The same is hardly true with HTML helpers.

View Components

Technically speaking, view components are self-contained components that include both logic and view. In ASP.NET Core, they replace child actions, which were available in classic ASP.NET MVC 5. You reference views components in Razor files via a C# block and pass them any input data that's required:

```
@await Component.InvokeAsync("LatestNews", new { count = 4 })
```

Internally, the view component will run its own logic, process the data you passed in and return a view ready for rendering. There are no predefined view components in ASP.NET Core, meaning that they'll be created on a strict application basis. The previous line of code presents a LatestNews view component that's a packaged block of markup conceptually similar to partial views. The difference between a partial view and a view component is in the internal implementation. A partial view is a plain Razor template that optionally receives input data and incorporates that in the HTML template of which it's made. A partial view isn't expected to have any behavior of its own except for formatting and rendering.

A view component is a more sophisticated form of a partial view. A view component optionally receives plain input parameters that it typically uses to retrieve and process its data. Once available, data is incorporated in an embedded Razor template, much the way a partial view behaves. However, a view component is faster in implementation because it doesn't go through the controller pipeline the way child actions do. This means there's no model binding and no action filters.

Overall, view components serve the noble purpose of helping to componentize the view so that it results from the composition of distinct and self-made widgets. This point is a double-edged sword. Having the view split into distinct and independent components seems a convenient way to organize the work and possibly make it more parallel by having different developers take care of the various parts. However, view components don't speed up things in all cases. It all depends on what each component does internally. When discussing view components, and the Composition UI pattern underneath its vision, you're often presented with the architecture of large service-oriented systems as an example, and that's absolutely correct.

However, when you blindly use view components, and the pattern, in the context of a monolithic, compact system, you might end up with poorly optimized query logic. Each component might query for its data, producing repeat queries and unnecessary database traffic. View components can be employed in this scenario, but you should consider caching layers to avoid bad outcomes.

Passing Data to Views

There are three different, non-exclusive ways to pass data to a Razor view (in addition to DI via the `@inject` directive). You can use one of the two built-in dictionaries—`ViewData` or `ViewBag`—or you can use strongly typed view model classes. No difference exists between these approaches from a purely functional point of view, and even from a performance perspective the difference is negligible.

From the design perspective, though, I recommend the single channel view model approach. Any Razor view should have a single source of data and that's the view model object. This approach demands that you pay close attention to the design of the base class for your view models, and that you avoid using dictionaries and even the `@inject` directive.

Any Razor view should have a single source of data and that's the view model object.

I realize that this approach sounds strict, but it comes from experience. Even calling `Date.Now` in the view can be dangerous, as it injects data into the view that might not be abstract enough for the application. `Date.Now` indicates the time of the server, not the time of the application. This is a problem for any application that depends on the time of the day for its operations. To avoid these kinds of traps, spend some time in the design of the view model classes to envision the common data you need to have in all views. Also, be sure to always pass data—any data—to the views via the model, avoiding dictionaries and DI as much as possible. While it's faster to use dictionaries and DI, it won't be faster later on when you have to refactor some domain functionality in a large application. DI in views is a cool piece of technology, but it should be used with more than just one grain of salt.

Wrapping Up

Views are the foundation of Web applications. In ASP.NET Core, views are the result of processing a template file—typically a Razor template file—that's mixed up with data provided by a caller that most often is a controller method. In this article, I tried to compare the various approaches available to build views and pass data to them. Most ASP.NET Core literature emphasizes the role of tag helpers and view components, and even Razor Pages and DI, in views. While these are all compelling resources, it's important not to use them blindly, as they have natural drawbacks that can produce some bad outcomes if you're not careful. ■

DINO ESPOSITO is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Programming ASP.NET Core" (Microsoft Press, 2018). A Pluralsight author and developer advocate at JetBrains, Esposito shares his vision of software on Twitter: @despos.

THANKS to the following Microsoft technical expert for reviewing this article: Steve Smith

msdnmagazine.com



Instantly Search Terabytes of Data

across a desktop, network, Internet or Intranet site with dtSearch enterprise and developer products

Over 25 search features, with **easy** **multicolor** **hit-highlighting** options

dtSearch's document filters support popular file types, emails with multilevel attachments, databases, web data

Developers:

- APIs for .NET, Java and C++
- SDKs for Windows, UWP, Linux, Mac and Android
- See dtSearch.com for articles on faceted search, advanced data classification, working with SQL, NoSQL & other DBs, MS Azure, etc.

Visit dtSearch.com for

- hundreds of reviews and case studies
- fully-functional evaluations

The Smart Choice for Text Retrieval®
since 1991

dtSearch.com 1-800-IT-FINDS



Kernel Logistic Regression Using C#

Kernel logistic regression (KLR) is a machine learning technique that can be used to make binary predictions. For example, KLR could predict if a person will repay a loan (fail to repay = 0, successfully repay = 1) based on predictor variables such as age, income and existing debt amount. KLR is an advanced variation of ordinary logistic regression.

A good way to see where this article is headed is to take a look at the demo program in **Figure 1** and the associated data in **Figure 2**. The goal of the demo program is to predict the class, 0 or 1, of dummy data that has just two predictor variables (sometimes called features), x_0 and x_1 . For example, the first training data item is (2.0, 3.0, 0), which means that if the predictor values are $x_0 = 2.0$ and $x_1 = 3.0$, the correct class is 0. KLR can handle data with any number of predictor variables, but using just two allows you to visualize the technique easily.

The 21 training data points have a circular geometry, which means that simple linear classification techniques, such as ordinary logistic regression, are ineffective. Such data is called non-linearly separable.

Behind the scenes, KLR uses a function called a radial basis function (RBF) kernel. The RBF kernel function has a parameter called sigma. The value of sigma must be determined by trial and error, and sigma is set to 1.0 in the demo. Training a KLR model is an iterative process and the demo sets the maximum number of iterations to 1,000, and sets a learning rate, eta, to 0.001.

Training a KLR model creates a set of “alpha values,” one for each training data item, plus an additional “bias” value. The demo program displays the alpha values for the first three training items (-0.3071, -0.3043, -0.3071) and the last two items (0.8999, 0.6108) and the bias (-1.0722).

After training, the KLR model predicts all 21 data items correctly. Then the model is applied to four test data items, shown as black dots in **Figure 2**. The first test item has inputs (1.5, 4.5) and a correct class of 0. The prediction model correctly predicts that item, and the other three test items, too.

This article assumes you have intermediate or higher programming skills but doesn't assume you know anything about KLR. The demo program is coded using C#, but you should have no trouble refactoring the code to another language, such as Java or Python, if you wish. The demo program is

too long to present in its entirety, but the complete source code is available in the file download that accompanies this article.

The RBF Kernel

A kernel function measures the similarity of two vectors or arrays. The RBF kernel function I mentioned earlier is the most common, and is the type used by the demo program. An RBF value of 1.0 means two vectors are identical. Smaller RBF values indicate two vectors are less similar.

The equation for RBF is:

$$K(v_1, v_2) = \exp(-\|v_1 - v_2\|^2 / (2 * \sigma^2))$$

Here, K stands for kernel; v_1 and v_2 are two vectors that have the same length; sigma is a parameter with a value like 1.0 or 1.5; the $\|$ indicates Euclidean distance; and the exp function is Euler's number ($e = 2.71828$) raised to a power.

```
file:///C:/KernelLogistic/bin/Debug/KernelLogistic.EXE
Begin kernel logistic regression demo
Goal is binary classification (0/1)
Setting up 21 training and 4 test data items
training [0] = (2.0, 3.0, 0)
training [1] = (1.0, 5.0, 0)
. . .
training [20] = (5.0, 6.0, 1)
Starting training
Using RBF kernel() with sigma = 1.0
Using SGD with eta = 0.001 and maxIter = 1000
Training complete
Trained model alpha values:
[0] -0.3071
[1] -0.3043
[2] -0.3071
. . .
[19] 0.8999
[20] 0.6108
[21] (bias) -1.0722
Evaluating model accuracy on train data
accuracy = 1.0000
Evaluating model accuracy on test data
input = (1.5, 4.5) actual = 0 calc y = 0.3049 pred = 0 correct
input = (7.0, 6.5) actual = 0 calc y = 0.2262 pred = 0 correct
input = (3.5, 4.5) actual = 1 calc y = 0.9272 pred = 1 correct
input = (5.5, 5.5) actual = 1 calc y = 0.6939 pred = 1 correct
numCorrect = 4 numWrong = 0
End kernel logistic regression demo
```

Figure 1 Kernel Logistic Regression Demo

Code download available at msdn.com/magazine/1117magcode.

The RBF function is best explained by example. Suppose $v1 = (3.0, 1.0, 2.0)$ and $v2 = (1.0, 0.0, 5.0)$, and σ is 1.5. First, you compute the squared Euclidean distance:

$$\begin{aligned} ||v1 - v2||^2 &= (3.0 - 1.0)^2 + (1.0 - 0.0)^2 + (2.0 - 5.0)^2 \\ &= 4.0 + 1.0 + 9.0 \\ &= 14.0 \end{aligned}$$

Next, you divide the squared distance by 2 times σ squared:

$$14.0 / (2 * (1.5)^2) = 14.0 / 4.5 = 3.11$$

Last, you take Euler's number and raise it to the negative of the previous result:

$$K(v1, v2) = e^{(-3.11)} = 0.0446$$

The small kernel value indicates that $v1$ and $v2$ are not very similar. The demo program defines the RBF kernel function as:

```
static double Kernel(double[] v1, double[] v2,
    double sigma)
{
    double num = 0.0;
    for (int i = 0; i < v1.Length - 1; ++i)
        num += (v1[i] - v2[i]) * (v1[i] - v2[i]);
    double denom = 2.0 * sigma * sigma;
    double z = num / denom;
    return Math.Exp(-z);
}
```

The function assumes that the last cell of each array holds the class label (0 or 1) and so the last cell isn't included in the calculation. KLR uses the kernel function to compare a given data item with all training items, and uses that information to determine a predicted class label.

Ordinary Logistic Regression

Ordinary logistic regression (LR) is best explained by example. Suppose you have three predictor variables: $x_0 = 2.5$, $x_1 = 1.7$, and $x_2 = 3.4$. A regular LR model creates a set of numeric constants called weights (w_i), one for each predictor variable, and an additional numeric constant called the bias (b). Note that the bias in regular LR isn't the same as the KLR bias shown in **Figure 1**.

The primary disadvantage
of regular LR is that it can
handle only data that's
linearly separable.

Suppose $w_0 = 0.11$, $w_1 = 0.33$, $w_2 = 0.22$, $b = 0.44$. To predict the class label, 0 or 1, for the input data (2.5, 1.7, 3.4) you first compute the sum of the products of each x and its associated w , and add the bias:

$$\begin{aligned} z &= (2.5)(0.11) + (1.7)(0.33) + (3.4)(0.22) + 0.44 \\ &= 2.024 \end{aligned}$$

Next, you compute $p = 1.0 / (1.0 + \exp(-z))$:

$$\begin{aligned} p &= 1.0 / (1.0 + \exp(-2.024)) \\ &= 0.8833 \end{aligned}$$

The p value is the probability that the data item has class label = 1, so if p is less than 0.5, your prediction is 0 and if p is greater than 0.5 (as it is in this example), your prediction is 1.

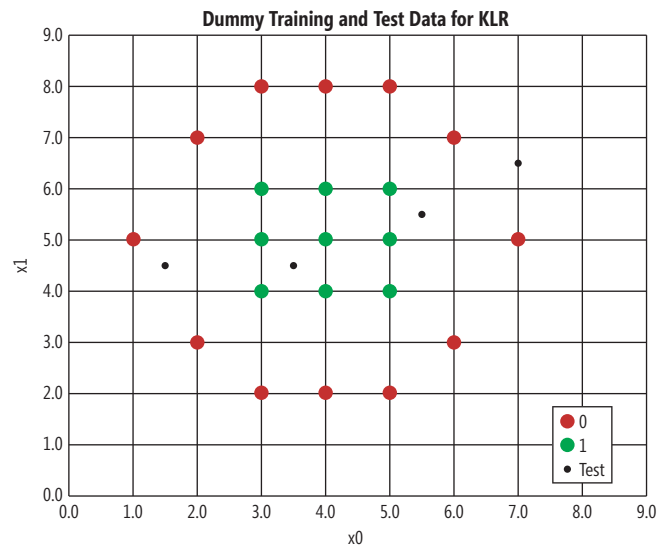


Figure 2 Kernel Logistic Regression Training Data

OK, but where do the weights and bias values come from in regular LR? The idea is that you determine the weights and bias values by using a set of training data that has known input values and known, correct class labels, then use an optimization algorithm to find values for the weights and biases so that the predicted class labels closely match the known, correct label values. There are many algorithms that can be used to find the weight and bias values for regular LR, including gradient ascent with log likelihood, gradient descent with squared error, iterated Newton-Raphson, simplex optimization, L-BFGS and particle swarm optimization.

The primary disadvantage of regular LR is that it can handle only data that's linearly separable. Regular LR can't handle data that's not linearly separable, such as the demo data shown in **Figure 2**.

Understanding Kernel Logistic Regression

KLR is best explained by example. Let me state up front that at first glance KLR doesn't appear very closely related to ordinary LR. However, the two techniques are closely related mathematically.

Suppose there are just four training data items:

```
td[0] = (2.0, 4.0, 0)
td[1] = (4.0, 1.0, 1)
td[2] = (5.0, 3.0, 0)
td[3] = (6.0, 7.0, 1)
```

Your goal is to predict the class label for $x = (3.0, 5.0)$. Suppose the trained KLR model gave you α values and a bias of: $\alpha[0] = -0.3$, $\alpha[1] = 0.4$, $\alpha[2] = -0.2$, $\alpha[3] = 0.6$, $b = 0.1$.

The first step is to compute the RBF similarity between the data item to predict each of the training items:

```
K(td[0], x) = 0.3679
K(td[1], x) = 0.0002
K(td[2], x) = 0.0183
K(td[3], x) = 0.0015
```

Notice that at this point, x is most similar to $td[0]$ and $td[2]$, which both have class label 0. Next, you compute the sum of products of each K value and the associated α , and add the bias value:

$$z = (0.3679)(-0.3) + (0.0002)(0.4) + (0.0183)(-0.2) + (0.0015)(0.6) + 0.1$$

$$= -0.1120$$

Now you compute $p = 1.0 / (1.0 + \exp(-z))$:

$$p = 1.0 / (1.0 + \exp(0.1120))$$

$$= 0.4720$$

If the p value is greater than 0.5, the predicted class is 1, and if the p value is less than 0.5, the predicted class is 0, as it is (just barely) for this example.

Training a KLR Model

Training a KLR model is the process of using training data to find the alpha values and the bias value. Expressed in very high-level pseudo-code, the KLR training algorithm is:

```
compute K(td[i], td[j]) for all i, j
loop maxIter times
  for-each curr training item, i
    for-each j: sum += alphas[j] * K(i, j)
    sum += bias
    y = 1.0 / (1.0 + exp(-sum))
    t = target class (0 or 1)
    for-each j:
      alpha[j] += eta * (t - y) * K(i, j)
    bias += eta * (t - y) * 1.0
end-loop
```

The key statement in the demo code is $\text{alphas}[j] += \text{eta} * (t - y) * \text{kernelMatrix}[i][j]$, which updates the alpha value for the training data item at index $[j]$ based on the current training data item at index $[i]$. Here, t is the known, correct target class, 0 or 1, and y is a calculated probability that the item at $[i]$ has class 1.

The Shuffle method is a helper that scrambles the order of the training items using the Fisher-Yates mini-algorithm.

For example, suppose an alpha value for a training item is currently 0.1234 and the target class is 1 and the computed probability is 0.60. The current prediction would be correct, but you'd like the p value to be even closer to 1. Suppose the similarity between the two items is $K(i, j) = 0.70$ and the learning rate eta is 0.10. The new alpha value would be:

$$\text{alpha} = 0.1234 + 0.10 * (1 - 0.60) * 0.70$$

$$= 0.1234 + 0.0280$$

$$= 0.1514$$

Because alpha is a multiplier value in the probability calculation, the new, slightly larger value of alpha will increase p a little bit, making the prediction more accurate.

The Demo Program

To code the demo program, I launched Visual Studio and created a new C# console application program and named it KernelLogistic. I used Visual Studio 2015, but the demo program has no significant .NET Framework dependencies, so any recent version of Visual Studio will work.

After the template code loaded into the editor window, I right-clicked on file Program.cs in the Solution Explorer window, renamed the file to KernelLogisticProgram.cs and then allowed Visual Studio to automatically rename class Program for me. At the top of the template-generated code, I deleted all unnecessary using statements leaving just the one that references the top-level System namespace. Then I instantiated a Random object:

```
using System;
namespace KernelLogistic
{
    class KernelLogisticProgram
    {
        static Random rnd = new Random(0);
        static void Main(string[] args)
        {
            Console.WriteLine("Begin KLR demo");
            int numFeatures = 2;
            ...
        }
    }
}
```

For simplicity, I coded the demo using a static method approach rather than object-oriented programming, and removed all normal error checking. The Main method sets up the 21 training items and the 4 test items like so:

```
double[][] trainData = new double[21][];
trainData[0] = new double[] { 2.0, 3.0, 0 };
...
trainData[20] = new double[] { 5.0, 6.0, 1 };
double[][] testData = new double[4][];
testData[0] = new double[] { 1.5, 4.5, 0 };
...
testData[3] = new double[] { 5.5, 5.5, 1 };
```

In a non-demo scenario, you'd likely read data from a text file. Next, the alpha values are initialized:

```
int numTrain = trainData.Length;
int numTest = testData.Length;
double[] alphas = new double[numTrain + 1];
for (int i = 0; i < alphas.Length; ++i)
    alphas[i] = 0.0;
```

When coding machine learning systems, there are usually several ways to deal with bias values. Here, I store the KLR bias in the last cell of the alphas array. An alternative design is to create a separate standalone variable. Next, the kernel similarities between all pairs of training items are computed:

```
double[][] kernelMatrix = new double[numTrain][];
for (int i = 0; i < kernelMatrix.Length; ++i)
    kernelMatrix[i] = new double[numTrain];
double sigma = 1.0;
for (int i = 0; i < numTrain; ++i) {
    for (int j = 0; j < numTrain; ++j) {
        double k = Kernel(trainData[i], trainData[j], sigma);
        kernelMatrix[i][j] = kernelMatrix[j][i] = k;
    }
}
```

Because there are only 21 data items, I sacrifice efficiency for simplicity. I could've reduced the number of kernel calculations by using the facts that $K(v_1, v_2) = K(v_2, v_1)$ and $K(v, v) = 1$. Next, the demo program prepares for training:

```
double eta = 0.001;
int iter = 0;
int maxIter = 1000;
int[] indices = new int[numTrain];
for (int i = 0; i < indices.Length; ++i)
    indices[i] = i;
```

The values of eta and maxIter were determined by trial and error. The idea behind the array named indices is that when training, it's important to visit the training items in a random order on each pass, to avoid getting into a situation where training stalls or oscillates back and forth. The main training loop begins:

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

1. Publication Title: *MSDN Magazine*
2. Publication Number: 1528-4859
3. Filing Date: September 30, 2017
4. Frequency of Issue: Monthly with a special issue in December.
5. Number of Issues Published Annually: 13
6. Annual Subscription Price: US \$35, International \$60
7. Complete Mailing Address of Known Office of Publication: 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311
8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher: Same as above.
9. Full Name and Complete Mailing Address of Publisher, Editor, and Managing Editor:
Henry Allain, President, 4 Venture, Suite 150, Irvine, CA 92618
Michael Desmond, Editor-in-Chief, 8251 Greensboro Drive, Suite 510, McLean, VA 22102
Wendy Hernandez, Group Managing Editor, 4 Venture, Ste. 150, Irvine, CA 92618
10. Owner(s): 1105 Media, Inc. dba: 101 Communications LLC, 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311. Listing of shareholders in 1105 Media, Inc.
11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or more of the Total Amount of Bonds, Mortgages or Other Securities:
Nautic Partners V, L.P., 50 Kennedy Plaza, 12th Fl., Providence, RI 02903
Kennedy Plaza Partners III, LLC, 50 Kennedy Plaza, 12th Fl., Providence, RI 02903
Alta Communications IX, L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
Alta Communications IX, B-L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
Alta Communications IX, Associates LLC, 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
12. The tax status has not changed during the preceding 12 months.
13. Publication Title: *MSDN Magazine*
14. Issue date for Circulation Data Below: September 2017
15. Extent & Nature of Circulation:

	Average No. Copies Each Month During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies (Net Press Run)	82,029	90,229
b. Legitimate Paid/and or Requested Distribution		
1. Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541	63,580	65,559
2. In-County Paid/Requested Mail Subscriptions Stated on PS Form 3541	0	0
3. Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS®	3,315	2,903
4. Requested Copies Distributed by Other Mail Classes Through the USPS®	0	0
c. Total Paid and/or Requested Circulation	66,895	68,462
Nonrequested Distribution		
1. Outside County Nonrequested Copies Stated on PS Form 3541	13,636	11,944
2. In-County Nonrequested Copies Distribution Stated on PS Form 3541	0	0
3. Nonrequested Copies Distribution Through the USPS by Other Classes of Mail	0	0
4. Nonrequested Copies Distributed Outside the Mail	1,383	9,708
e. Total Nonrequested Distribution	15,019	21,652
f. Total Distribution	81,914	90,114
g. Copies not Distributed	115	115
h. Total	82,029	90,229
i. Percent paid and/or Requested Circulation	81.66%	75.97%

16. Electronic Copy Circulation
 - a. Requested and Paid Electronic Copies
 - b. Total Requested and Paid Print Copies (Line 15c) + Requested/Paid Electronic Copies
 - c. Total Requested Copy Distribution (Line 15f) + Requested/Paid Electronic Copies (Line 16a)
 - d. Percent Paid and/or Requested Circulation (Both print & Electronic Copies) (16b divided by 16c x 100)

☒ I certify that 50% of all my distributed copies (electronic and paid print are legitimate request or paid copies.
17. Publication of Statement of Ownership for a Requester Publication is required and will be printed in the November 2017 issue of this publication.
18. I certify that all information furnished on this form is true and complete:
Peter B. Weller, Manager, Print Production

```
while (iter < maxIter) {
    Shuffle(indices);
    for (int idx = 0; idx < indices.Length; ++idx) {
        int i = indices[idx];
```

The Shuffle method is a helper that scrambles the order of the training items using the Fisher-Yates mini-algorithm. The target class label and the predicted probability of the current training item are computed like so:

```
double sum = 0.0;
for (int j = 0; j < alphas.Length-1; ++j)
    sum += alphas[j] * kernelMatrix[i][j];
sum += alphas[alphas.Length - 1];
double y = 1.0 / (1.0 + Math.Exp(-sum));
double t = trainData[i][numFeatures];
```

Notice that this design assumes that the class label is in the last cell of a training data array. Next, the alphas and the beta values are updated:

```
for (int j = 0; j < alphas.Length - 1; ++j)
    alphas[j] = alphas[j] +
        (eta * (t - y) * kernelMatrix[i][j]);
alphas[alphas.Length-1] = alphas[alphas.Length - 1] +
    (eta * (t - y)) * 1;
}
++iter;
} // While (train)
```

Updating the bias value uses a dummy value of 1 in place of the kernel similarity value, just to make the symmetry of the relationship clear. Of course, you can remove the multiplication by 1 because it has no effect. After training, a few of the values of the alphas and the bias value, are displayed, as shown in **Figure 1**.

The demo program concludes by computing the classification accuracy of the trained KLR model on the training and test data:

```
double accTrain = Accuracy(trainData, trainData,
    alphas, sigma, false);
Console.WriteLine("accuracy = " +
    accTrain.ToString("F4") + "\n");
double accTest = Accuracy(testData, trainData,
    alphas, sigma, true); // Verbose
```

The Boolean argument passed to method Accuracy indicates whether to compute in verbose mode (with diagnostic messages) or silent mode.

Wrapping Up

Kernel logistic regression isn't used very often, at least among my colleagues. Its major advantage is simplicity. The major disadvantage of KLR is that it doesn't scale well to large data sets because you either have to precompute all item-to-item kernel similarity values and save them, or you must keep all training data and then compute all similarity values on the fly for every prediction.

KLR is designed for binary classification. It's possible to extend KLR to handle classification problems with three or more class values, but in my opinion, there are better alternatives to use, in particular a single hidden layer feed-forward neural network. KLR has some similarities to the K nearest neighbors (K-NN) classification algorithm, and also to support vector machine (SVM) classification. ■

Dr. James McCaffrey works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.

THANKS to the following Microsoft technical experts who reviewed this article:
Chris Lee and Adith Swaminathan

VISUAL STUDIO LIVE! (VSLive!™) is celebrating 25 years as one of the most respected, longest-standing, independent developer conferences, and we want you to be a part of it.

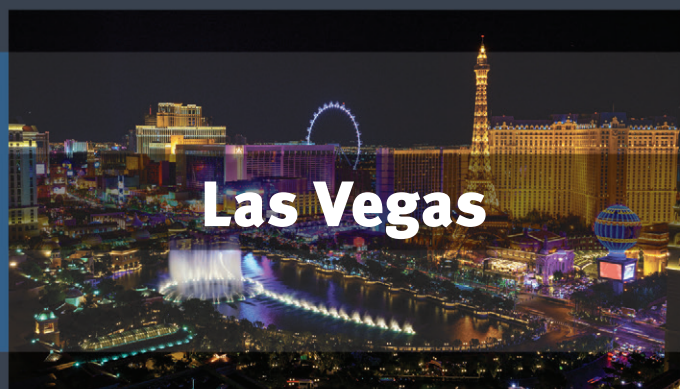
Join us in 2018 for #VSLive25, as we highlight how far technology has come in 25 years, while looking toward the future with our unique brand of training on .NET, the Microsoft Platform and open source technologies in seven great cities across the US.

March 12 – 16, 2018

Bally's Hotel & Casino



Respect the Past.
Code the Future.



April 30 – May 4, 2018

Hyatt Regency Austin



Code Like It's 2018!



June 10 – 14, 2018

Hyatt Regency Cambridge



Developing Perspective.



SUPPORTED BY



Visual Studio
MAGAZINE

PRODUCED BY



August 13 – 17, 2018

Microsoft Headquarters



Yesterday's Knowledge;
Tomorrow's Code!



September 17 – 20, 2018

Renaissance Chicago



Look Back to
Code Forward.



NEW LOCATION!

October 8 – 11, 2018

Hilton San Diego Resort



Code Again for
the First Time!



December 2 – 7, 2018

Loews Royal Pacific Resort



Code Odyssey.



CONNECT WITH US



twitter.com/vslive –
@VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!

vslive.com

#VSLIVE25



Duct Tape

Duct tape. *Duct* tape. Where would this world be without duct tape? Its name implies usage on ducts, but I've never actually seen that. I have, at one time or another, seen duct tape plastered onto almost every object in the universe that I've ever observed. Fans of the "Red Green Show" (redgreen.com) know it as the handyman's all-purpose secret weapon. The Apollo 13 astronauts used it to construct the air filters that saved their lives on their way back from the moon (go.nasa.gov/2yr9N7I). In Andy Weir's superb science fiction novel, "The Martian," astronaut Mark Watney uses it again and again to save his life while stranded on Mars: "Yes, of course duct tape works in a near-vacuum. Duct tape works anywhere. Duct tape is magic and should be worshipped."

Yet duct tape has a darker side: the reputation of being used for unworthy hacks and kludges. "Held together with duct tape" is a pejorative phrase. The strong aluminum speed tape used for temporary repairs on airplanes (bit.ly/2wis5aQ) resembles duct tape when seen from a distance, causing passengers to freak out: "There's duct tape on my plane!" (bit.ly/2yExbZC). And once a duct-taped hack is in place, it's almost impossible to get rid of.

All of the mechanisms we use to
organize our software world—
functions, objects, services,
clouds, all of them—fall helpless
before the entropy of GOTO.
And yet, we can't get rid of it.

The concept of duct tape has wormed its way into our collective consciousness, even for non-tangible situations. Jim Morris sang that "Booze is the duct tape of life." So, my geeky friends, what is the duct tape of software? What keeps patched-up Windows XP systems running today? What keeps critical applications alive long after all of their programmers have died?

One friend suggested that Google is the duct tape of software. Can't remember something? Just Google it. The term entered the language more than a decade ago (bit.ly/2ftqCYf).

Certainly search engines, coupled with the ubiquity of smart devices, have created a world where any question of fact can be answered instantaneously. It's an interesting foundation for a new

social order, but it hasn't been around long enough to hold the software world together.

Perhaps it's StackOverflow. Whenever I get an exception I don't understand, I Google it, and there it is on StackOverflow. No one could program today without StackOverflow or something similar; the programming world is just too damn complicated. But even StackOverflow doesn't qualify as the binding energy of the software universe.

So what now? The pipe character? Shell scripts? Environment variables? VB6?

After extensive research, I've concluded that the duct tape of software is the infinitely abusable GOTO statement. The mere mention evokes fervent emotions both for and against its use. Edsger Dijkstra fulminated against it (bit.ly/2fwLuBU), and Frank Rubin fulminated against Dijkstra's fulminations (bit.ly/2y8Y3uu). I'm sure you just did a double-take yourself, probably with a grimace, wondering, "Did Plattski actually write *the G word*?"

All the mechanisms we use to organize our software world—functions, objects, services, clouds, all of them—fall helpless before the entropy of GOTO. And yet, we can't get rid of it. From the first BASIC compiler I used in school back in 1975, through today's C# and Java, every tool contains GOTO. When a language developer tries to be virtuous by omitting it, some antisocial geek immediately adds it, as with Ruby (bit.ly/2yF9W8M), JavaScript (bit.ly/2xv2b7s) and Python (bit.ly/2xuMYU0)—the last allegedly an April Fools' Day joke (hooray!), but it still works.

You can prove to me that GOTO isn't necessary, that all Turing functions can be accomplished without it. I'll even admit that you're right. But as we learned from the Y2K kerfuffle, the continuing function of society depends on the continued operation of piles of code that no programmer now living has ever laid eyes on. It's often full of GOTOs. You can't get rid of these old systems, as they do indeed hold the world together. And it's almost impossible to seriously refactor them. We're stuck. So I hereby declare GOTO, that unkillable cockroach, to be the duct tape of software. Enjoy it.

One of these days I'll write the complement to this article, a discussion of duct tape's anti-particle, the positron to duct tape's electron. Which is, of course, WD-40. ■

DAVID S. PLATT teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.



Looking for a powerful distributed cache?

Get the easiest, most powerful in-memory data grid designed to scale your .NET applications.

- ▶ Fast and linearly scalable
- ▶ Enterprise-grade availability
- ▶ Industry-leading ease of use
- ▶ Integrated in-memory computing

Go with the industry leader in distributed caching

Trusted by hundreds of enterprise customers for more than 12 years, ScaleOut's distributed caching technology delivers rock-solid performance, legendary ease of use, and world-class support. Unlike Redis, it offers a better migration path from AppFabric Caching. Here's why:

	ScaleOut	Redis
Source-code compatible AppFabric Caching APIs for seamless migration	✓	✗
Architected from the ground up for automatic scaling and high availability	✓	✗
Fully coherent data storage and access for mission-critical applications	✓	✗
Advanced .NET features: distributed LINQ query, C# MapReduce, and more	✓	✗

Replacing AppFabric Caching? Trouble switching to Redis?

Check out our AppFabric-compatible drop-in: www.scaleoutsoftware.com/appfabric



ScaleOut Software



KEEP CALM AND STOP COUNTING USERS

Why pay for expensive software development costs
when you can have it all with one low, annual fee!

- No user count
- Access to 800+ pre-built controls and frameworks
- Plus Big Data, Dashboard, Data Integration, and Report platforms
- Truly unlimited use
- Starting at \$3,995

START YOUR EVALUATION TODAY!

www.syncfusion.com/MSDNunlimited

